The groovy programming language

Fred Mora – Nov 2014

What is Groovy?

- Based on Java, without the boilerplate code
- Perfect for scripts, test programs, system "glue" code
- Often described as "Perl for Java"
- Natively executes any Java API
- Can be used in simple scripts or in complex, multi-classes OO programs
- If you use, test or write Java, you can probably benefit immediately from using Groovy
- If you write or maintain shell scripts or build scripts, you can benefit too!

The Hello World code

- Groovy code can be used in OO programs...

```
class Hello {

    public static void main(String[] args) {
        println "Hello, World!"
    }
}

=> Hello, World!
```

- ...Or in "classless" scripts!

```
println "Hello, World!"

=> Hello, World!
```

A few differences

- In a Groovy method:
  - Attributes are private by default
  - Methods are public by default
  - Getters and setters are auto-generated
- To invoke the class:
  - You can use a default constructor (more later)
  - You can say instance.field instead of getField() and setField(). Example:

```
universe.secret = 42
println universe.secret
```

Easy-to-use Lists and Hash

- Lists and hashes can be defined directly

```
cities = [ "New York", "Chicago" ]
citySlogans = [ "New York": "Big Apple", "Chicago": "Windy City" ]
starWarsCity = ["Mos Eisley": "Wretched hive of scum and villainy" ]
citySlogans += starWarsCity
println citySlogans

=> [New York:Big Apple, Chicago:Windy City, Mos Eisley:Wretched hive
of scum and villainy]
```

•Note the append operation for hashes
•Most Groovy objects try to have a printable toString().

Closures

- Closures are anonymous, inner methods
- Many standard Groovy methods use closures as arguments.

```
cities = [ "New York", "Chicago", "Mos Eisley" ]
compNameCities = cities.findAll { name -> name.find(" ")}
println "These cities have a composite name: " + compNameCities.join(", ")

=> These cities have a composite name: New York, Mos Eisley
```

- Closures can take several arguments:

```
println "The same as a nice list:"
compNameCities.eachWithIndex{ name, i -> println "${i + 1}. $name" }

=> The same as a nice list:
1. New York
2. Mos Eisley
```

Notice the expression embedded in the string, like in Perl! Groovy calls these composite strings "GStrings" (roll eyes).

Closures as shorthand notations

- Basic loops

```
5.times{print "*"}
=> *****

(1..10).each{print "$it "}
 => 1 2 3 4 5 6 7 8 9 10
```

- Hash member access and search

```
starWars = ['Quotes' : [ 'Obi-wan': 'Use the force, Luke',
                         'Yoda': 'You must unlearn what you have learned.',
                         'Han Solo': "Don't get cocky" ],
            'Actors' : [ 'Mark Hamill': 'Luke Skywalker',
                         'Harrison Ford': 'Han Solo'] ]

assert starWars['Quotes'][ 'Obi-wan'] == 'Use the force, Luke'
// Shorthand
assert starWars.Quotes.'Obi-wan' == 'Use the force, Luke'
// Lookup in a hash
someCharacter = starWars.Quotes.find{it.value == "Don't get cocky"}.key
assert starWars.Actors.find{it.value == someCharacter}.key ==  'Harrison Ford'
```

Closures and lists

- The Groovy way of iterating through a list is to use a method and a closure, *not* a for loop.

```
cities = [ "New York", "Chicago", "Mos Eisley" ]
cities.each{ println "Length of the string $it: ${it.size()}" }

// Get the sizes of these strings in a list
def sizes = []
cities.each{ sizes << it.size() }
println sizes

=>
Length of the string New York: 8
Length of the string Chicago: 7
Length of the string Mos Eisley: 10
[8, 7, 10]
```

- This is common enough that Groovy offers the star-dot operator as a shorthand:

```
// Shorten this with star-dot operator
assert cities*.size() == [8, 7, 10]
```

Closures and lists (cont'd)

- Make a collection based on a criterion

```
println cities.findAll { it.size() >= 8 }
=>
[New York, Mos Eisley]
```

- Count elements that match a criterion - Here, criterion = "is multiple of ten"

```
search = ((1..1000).countBy { it %10 == 0 })
println search
assert search[(true)] == 100
=>
[false:900, true:100]
```

Closures as fields (a.k.a. Expandos)

- Expandos are objects with computed fields that are closures. They are like methods in classes, but they work in scripts.

```
final ipToLoc = [ "10.204": "FFD", "10.205": "MH", "10.18": "NDA",
"10.19": "BLR" ]

def machine = new Expando()
machine.address = "10.204.23.53"
machine.location = {ipToLoc.find{address.startsWith(it.key)}.value}

println machine.address + " is in " + machine.location()
machine.address = "10.205.78.56"
println machine.address + " is in " + machine.location()

=>

10.204.23.53 is in FFD
10.205.78.56 is in MH
```

Groovy knows about Java packages

- Many standard Java packages are included out of the box

```
classes = [File, HashMap, List, Map, String]
for (c in classes) {
  println "The class " + c.simpleName + " is in package "
    + c.package.name
}

=>
The class File is in package java.io
The class HashMap is in package java.util
The class List is in package java.util
The class Map is in package java.util
The class String is in package java.lang
```

Groovifying the Java code

- The previous code is straight Java without semicolons (and System.out, and classes) but we can still remove more boilerplate.

```
[File, HashMap, List, Map, String].each { c->
  println "The class ${c.simpleName} is in package ${c.package.name}"
}

=>
The class File is in package java.io
The class HashMap is in package java.util
The class List is in package java.util
The class Map is in package java.util
The class String is in package java.lang
```

A class example showing overloading and implicit constructors

```
class Book {
    List<Location> places = []
    List<Persona> characters = []

    Book leftShift(final Location loc) {
        places << loc
        this  // Implicitly returns the last expression of a block
    }

    Book leftShift(final Persona pers) {
        characters << pers
        this
    }
}
class Location {
     String name
}
class Persona {
     String name
     String description
}

final Book lotr = new Book()
lotr << new Persona(name: 'Bilbo', description: 'Hobbit') << new Location(name: 'Shire')
lotr << new Persona(name: 'Gandalf',  description: 'Wizard')

assert lotr.places.size() == 1
assert lotr.characters.size() == 2
```

Adding automatic cache to you code

- Often, when dealing with long computation or slow external systems, we add some caching to increase performance
- Groovy offers in-memory caching. When methods or closures are called multiple times with the same args, the result is pulled from the cache starting from the 2nd time.

```
def MassiveCalculation = {a, b -> sleep 2000; a+b}.memoize()
def Tprintln = {it -> print new Date(); println " - " + it }

Tprintln ("Starting computation")
Tprintln ("Result for 2 and 3: " + MassiveCalculation(2, 3))
Tprintln ("Result for 4 and 5: " + MassiveCalculation(4, 5))
Tprintln ("Result for 2 and 3 again: " + MassiveCalculation(2, 3))
Tprintln ("Result for 4 and 5 again: " + MassiveCalculation(4,5))

=>
Thu Jul 05 18:32:30 EDT 2012 - Starting computation
Thu Jul 05 18:32:32 EDT 2012 - Result for 2 and 3: 5
Thu Jul 05 18:32:34 EDT 2012 - Result for 4 and 5: 9
Thu Jul 05 18:32:34 EDT 2012 - Result for 2 and 3 again: 5
Thu Jul 05 18:32:34 EDT 2012 - Result for 4 and 5 again: 9
```

Consuming structured data

- Groovy can read complex data format such as JSON and XML

```
// Grab the first 3 cities within certain long. and lat.
import groovy.json.*
String url='http://api.geonames.org/citiesJSON?
north=41.1&south=41.5&east=-72.5&west=-73&maxRows=3&username=demo'
def data = new URL(url).text
def doc = (new JsonSlurper()).parseText(data)
doc.geonames.each {
  println "Town: ${it.name}, population: ${it.population}"
}

=>
Town: New Haven, population: 129779
Town: Hamden, population: 59847
Town: West Haven, population: 55564

// Json text looks like:
// {"geonames":[{""name":"New Haven","countrycode":"US","population":129779,...}...}]}
```

Producing structured data

- Groovy offers builders for XML and JSON

```
import groovy.json.*
def json = new JsonBuilder()

json.company {
   name "Foomatic"
  products
    { name "Better Mouse Trap"; versions 1, 2 }
    { name "Universal Bug Remover"; versions 14, 15, 16 }
}
println JsonOutput.prettyPrint(json.toString())
=>
{
    "company": {
        "name": "Foomatic",
        "products": [
            {
                "name": "Better Mouse Trap",
                "versions": [
                    1,
                    2
                ]
            },
...
```

Annotations

- Groovy compiles to bytecode
- Some annotations inject code in the Abstract Syntax Tree stage of the compilation
- Hence the name "AST Transforms" for this category of annotations
- Example:
  - @ToString – Auto-creates toString() method
  - @TupleConstructor: tuple-based constructor.

```
import groovy.transform.*

@ToString(includeNames = true)
@TupleConstructor
class Persona {
     String name, description
     int age
}

//Normal implicit constructor vs. tuple constructor
def w1 = new Persona(name: 'Gandalf',  description: 'Wizard', age: 2000)
def w2 = new Persona('Gandalf', 'Wizard', 2000)
println w1
=>
Persona(name:Gandalf, description:Wizard, age:2000)
```

Annotations: Logging

- @Log (and @Log4j): Log calls with auto-generated "if" guard.

```
import groovy.util.logging.*
// Injects a java.util.logging Logger. Available levels:
// finest (lowest), finer, fine, config, info, warning, severe (highest)
// Default is info
@Log
class Country {
  String name
  String capital
  Country(n, c) {
    name = n;capital = c
    log.info 'In Country constructor'
    // Implicit 'if' guard
    log.finest ("Exception bait: " + 1/0)
  }
}

def fr = new Country('France', 'Paris')
=>
Jul 6, 2012 7:15:37 PM java_util_logging_Logger$info call
INFO: In Country constructor
```

Annotations: Interrupts and Timers

- Timed interrupt created automatically

```
@TimedInterrupt(10) // 10 seconds
import groovy.transform.TimedInterrupt

try {
  while (true) {
      // Wait on some other threads
  }
}
catch(java.util.concurrent.TimeoutException e) {
  println "Got interrupted"
}

=>

Got interrupted
```

File access with Groovy

- It's easy to access the disk and look for files

```
dir = new File(path).eachFileRecurse{ f->
    print f
    if (f.isFile()) { println " (${f.size()} bytes)" }
    if (f.isDirectory()) { println "/" }
  }
}

showContents( "TimePatrol")

=>

TimePatrol/flux-capacitor-users-manual.pdf (2189567 bytes)
TimePatrol/trip-reports/
TimePatrol/trip-reports/1215-magna-carta-written.pdf (15490 bytes)
TimePatrol/trip-reports/1571-battle-of-lepante.pdf (21927 bytes)
TimePatrol/trip-reports/1804-napoleon-crowned-emperor.pdf (19855 bytes)
TimePatrol/trip-reports/2021-dogs-get-person-status.pdf (1162336 bytes)
```

Looking at file contents with regular expressions

```
// Create the test file
file = new File("BTTFdata.txt")
file << '''Marty: Wait a minute, Doc... Are you telling me
that you built a time machine... out of a DeLorean?
Doc Brown: The way I see it, if you're gonna build a
time machine into a car, why not do it with some style?
'''

// Now find all the files with a name starting with "BTTF"
def filter = { dir, name -> name.startsWith("BTTF") } as FilenameFilter
def matchingFiles = new File('.').list (filter)

// Take the first file found, look for regex
file = new File(matchingFiles[0])
file.eachLine { line ->
  regex = '''(?x:  # Header for an extended regex
             \\W     # Non-word char
             [a-z]*  # 0 or more lowercase
             [A-Z]   # 1 cap
             [a-z]*  # 0 or more lowercase
             [A-Z]   # 1 cap
             )'''
   if (line =~ regex) { println "Line with a 2-cap word: $line"}
}
=>
Line with a 2-cap word: that you built a time machine... out of a DeLorean?
```

Running system commands

- Portability is for wimps. Let's get down to the shell!

```
// Simple command
cmd1 = "df  -h ."
def process = cmd1.execute()
println "Output of \"$cmd1\" is:\n${process.text}"

cmd2 = "tail -1"
println "Now running command $cmd1 | $cmd2"
process1 = cmd1.execute()
process2 = cmd2.execute()
process1 | process2
process2.waitFor()
if (process2.exitValue())
    print "Error:" + process2.err.text
else
    print "Output:" + process2.text
=>
Output of "df  -h ." is:
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda3             438G  103G  314G  25% /home

Now running command df  -h . | tail -1
Output:/dev/sda3             438G  103G  314G  25% /home
```

Simplifying Ant scripts with Groovy

- Ant has many advantages (I'm sure).
- Concise syntax is not one of them. Some sample Ant code that check properties:

```
<!-- Load the Ant contribs for the if task -->
<taskdef resource="net/sf/antcontrib/antlib.xml">
  <classpath>
    <pathelement location="${buildtools.dir}/sf.net/ant-contrib_0.3/ant-contrib-1.0b3.jar" />
  </classpath>
</taskdef>

<!-- Code for deciding if the build output should be sent to the NAS -->
<if>
  <and>
    <istrue value="${publishOutput}" />
    <or>
      <istrue value="${publishOutput.onFailure}" />
      <equals arg1="${publishOutput.build.status}" arg2="SUCCEEDED" />
    </or>
  </and>
  <then>
    <antcall target="-execPublishOutput" />
  </then>
  <else>
    <echo>Output publication skipped. Either publishOutput is not set to true,
      or the build failed and publishOutput.onFailure is false.
    </echo>
  </else>
</if>
```

Simplifying Ant scripts with Groovy (cont'd)

- Let's use the groovy task, which provides access to ant tasks and to the caller's properties.

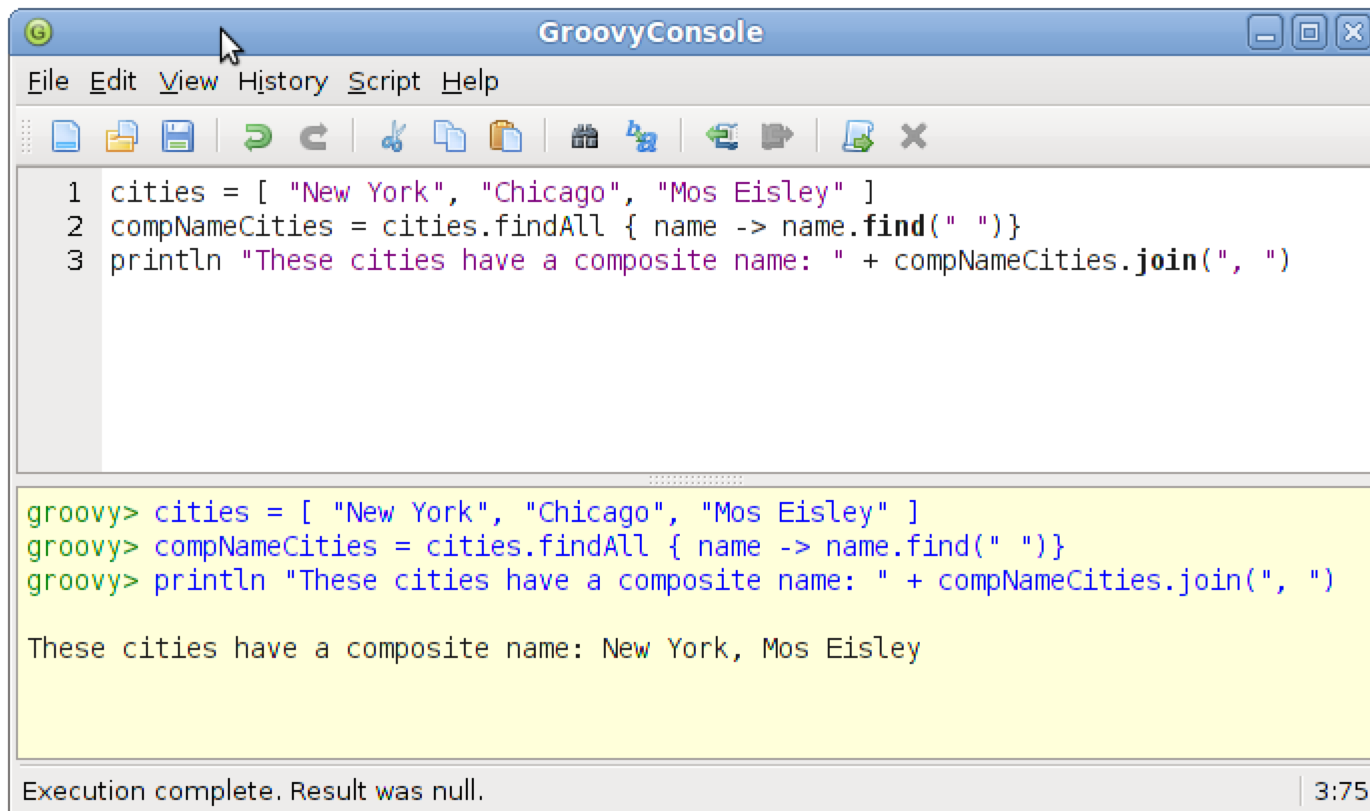
```
<taskdef name="groovy"
          classname="org.codehaus.groovy.ant.Groovy"
          classpath="$
{buildroot.dir}/3rdparty/groovy/groovy-1.8.4/groovy-all-1.8.4.jar"/>

<groovy>
if ( (properties.'publishOutput'.toLowerCase() == "true") &&
     ( ( properties.'publishOutput.onFailure'.toLowerCase() == "true" ) ||
       ( properties.'publishOutput.build.status' == "SUCCEEDED" )
     )
   ) {
  ant.antcall(target: "-execPublishOutput")
} else {
  ant.echo("Output publication skipped. Either publishOutput is not set to
true, or the build failed and publishOutput.onFailure is false."
}
</groovy>
```

Running Groovy

- Install from groovy.codehaus.org
- Start your scripts with the hashbang
  #! /usr/bin/groovy
  or with the more universal
  #! /usr/bin/env groovy
  which simply assumes groovy is in your path
- Install the Eclipse Groovy plugin if needed
- Multi-file packages get compiled in Eclipse. You'll need a build script.

The Groovy Console

- Use the Groovy Console to test and debug snippets
- Command: groovyConsole
- Lets the user examine objects



That's all folks...