Get Groovy

I dare you to.

Code available at

https://github.com/jkelly467/cvcc14_groovy

History

- [ ] James Strachan dreamed up Groovy in 2003
- [ ] Version 1.0 released in 2007
- [ ] Won the JAX 2007 innovation award

Main Concepts

- ☐ Agile programming for the JVM
- ☐ Designed for ease of use and quick iteration
- ☐ Syntax and features brought in from Ruby and Python (among others)

Examples

Dynamic Typing

- Obviously, Java requires types to be defined explicitly at the compilation step
- In Groovy, an object's type is discovered dynamically at runtime

Java

```java
String aStr = "A string";
System.out.println(aStr); //A string
Integer anInt = 1;
System.out.println(anInt); //1
```

Groovy

```groovy
aVar = "A string"
println aVar //A string
aVar = 1
println aVar //1
```

More Dynamic Typing

- We don’t need to explicitly declare types when defining class variables or function parameters either
- This can give us polymorphism without inheritance, and save us a bunch of code

Java

```java
public class NamedThing {
    protected String name;

    public NamedThing(String name) {
        this.name = name;
    }

    public String getName() { return name; }
}
```

```java
import java.math.BigDecimal;

public class Song extends NamedThing{
    private BigDecimal length;

    public Song(String name, BigDecimal length) {
        super(name);
        this.length = length;
    }

    public BigDecimal getLength() { return length; }
}
```

```java
public class Book extends NamedThing{
    private String author;

    public Book(String name, String author) {
        super(name);
        this.author = author;
    }

    public String getAuthor() { return author; }
}
```

```java
import java.math.BigDecimal;

public class Test {

    private static void printName(NamedThing thing) {
        System.out.println(thing.getName());
    }

    public static void main(String[] args) {
        printName(new Song("ABCs", new BigDecimal(1.0))); //ABCs
        printName(new Book("Game of Thrones", "George R. R. Martin")); //Game of Thrones
    }

}
```

Groovy

```groovy
class Song (
  name
  length
)

class Book (
  name
  author
)

void doSomething(thing) {
  println thing.name
}

doSomething(new Song(name: "ABCs", length:1.0))
doSomething(new Book(name: "Game of Thrones", author: "George R. R. Martin"))
```

```
jkelly@ec-devcave9 groovy/dynamic_typing » groovy Polymorphism.groovy
ABCs
Game of Thrones
```

Dynamic Typing - def

```
def action(param) {
  param = param + 6
  return param
}

def test = "Hello"
def result1 = action(test)

assert result1 == "Hello6"

test = 6
def result2 = action(test)

assert result2 == 12
```

Closures

- A code block, similar to anonymous inner classes in Java
- Code that is defined and run later
- Java 8 has a lot of this

Java

```java
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Test {

    public static void main(String[] args) {
        List<String> list = Arrays.asList(new String[]{"superlonglong superlonglong", "short", "longlonglong", "medium"});
        List<String> list2 = Arrays.asList(new String[]{"a", "aaaa", "aaa", "aa"});

        Collections.sort(list, new Comparator<String>(){

            @Override
            public int compare(String o1, String o2) {
                return new Integer(o1.length()).compareTo(o2.length());
            }

        });

        Collections.sort(list2, new Comparator<String>(){

            @Override
            public int compare(String o1, String o2) {
                return new Integer(o1.length()).compareTo(o2.length());
            }

        });

        System.out.println(list); //[short, medium, longlonglong, superlonglong superlonglong]
        System.out.println(list2); //[a, aa, aaa, aaaa]
    }

}
```

Groovy

```groovy
def list = ["superlonglong superlonglong", "short", "longlonglong", "medium"]
def list2 = ["a", "aaaa", "aaa", "aa"]

list.sort { a, b ->
  a.length().compareTo(b.length())
}

list2.sort { a, b ->
  a.length().compareTo(b.length())
}

println "List: "+list
println "List2: "+list2
```

```
jkelly@ec-devcave9 groovy/closures » groovy Sort1.groovy
List: [short, medium, longlonglong, superlonglong superlonglong]
List2: [a, aa, aaa, aaaa]
```

Java

```java
import java.util.Comparator;

public class StringLengthComparator implements Comparator<String> {

    @Override
    public int compare(String o1, String o2) {
        return new Integer(o1.length()).compareTo(o2.length());
    }

}
```

```java
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class Test {

    public static void main(String[] args) {
        List<String> list = Arrays.asList(new String[]{"superlonglong superlonglong", "short", "longlonglong", "medium"});
        List<String> list2 = Arrays.asList(new String[]{"a", "aaaa", "aaa", "aa"});

        Collections.sort(list, new StringLengthComparator());

        Collections.sort(list2, new StringLengthComparator());

        System.out.println(list); //[short, medium, longlonglong, superlonglong superlonglong]
        System.out.println(list2); //[a, aa, aaa, aaaa]
    }

}
```

Groovy

```groovy
def list = ["superlonglong superlonglong", "short", "longlonglong", "medium"]
def list2 = ["a", "aaaa", "aaa", "aa"]

def comparator = { a, b ->
  a.length().compareTo(b.length())
}

list.sort(comparator)
list2.sort(comparator)

println "List: "+list
println "List2: "+list2
```

```
jkelly@ec-devcave9 groovy/closures » groovy Sort2.groovy
List: [short, medium, longlonglong, superlonglong superlonglong]
List2: [a, aa, aaa, aaaa]
```

Groovy?

```groovy
def list = ["superlonglong superlonglong", "short", "longlonglong", "medium"]
def list2 = ["a", "aaaa", "aaa", "aa"]

def arbitrary = 10

def comparator = { a, b ->
  (a.length() + b.length()).compareTo(arbitrary)
}

list.sort(comparator)
list2.sort(comparator)

println "List: "+list
println "List2: "+list2
```

```
jkelly@ec-devcave9 groovy/closures » groovy Sort3.groovy
List: [superlonglong superlonglong, short, longlonglong, medium]
List2: [aa, aaa, aaaa, a]
```

Functional - each

```groovy
def list = ["first", "second", "third", "fourth"]

list.each {
  println it
}
```

```
jkelly@ec-devcave9 groovy/closures »  groovy Each.groovy
first
second
third
fourth
```

Functional - collect

```
def list = ["first", "second", "third", "fourth"]

def newList = list.collect {
  return it+" item"
}

println newList
```

```
jkelly@ec-devcave9 groovy/closures » groovy Collect.groovy
[first item, second item, third item, fourth item]
```

Functional - find

```groovy
def MASTER = ["foo", "bar"]
def list = ["foo", "baz", "bar", "brah", "breh", "bluh"]

println list.find {
  return MASTER.contains(it)
}
```

```
jkelly@ec-devcave9 groovy/closures » groovy Find.groovy
foo
```

Functional - findAll

```groovy
def MASTER = ["foo", "bar"]
def list = ["foo", "baz", "bar", "brah", "breh", "bluh"]

println list.findAll {
  return MASTER.contains(it)
}
```

```
jkelly@ec-devcave9 groovy/closures »  groovy FindAll.groovy
[foo, bar]
```

Sweet Sweet Groovy

Some nice stuff to make your life easier

Strings

- ☐ Java has one way to declare Strings. That's boring
- ☐ Groovy has 3!

```
assert 'test' == "test"

assert "test line 1\n test line 2" ==
"""test line 1
 test line 2"""
```

Groovy Strings

- The, erm, GString. Allows templating style when creating strings
- Anything enclosed by ${} in a double quoted string will be evaluated as a Groovy expression
- This includes Closures

```groovy
class Me {
  def first = "James"
  def last = "Bond"

  def fullName() {
    return first + " " + last
  }

  String toString() {
    return last
  }
}

def me = new Me()

println "The name's $me, ${me.fullName()}"
println "The name's $me.last, $me.first $me.last"
timeGuy = "Also, did you know that ${new Date()} is before ${writer -> writer << new Date()}"
sleep 1000
println timeGuy
```

```
jkelly@ec-devcave9 groovy/strings » groovy GString.groovy
The name's Bond, James Bond
The name's Bond, James Bond
Also, did you know that Tue Oct 21 10:55:59 CDT 2014 is before Tue Oct 21 10:56:00 CDT 2014
```

Implicit Return

- The last statement in a method or closure will be returned

```groovy
def aWeirdName() {
  "Jon Kelly"
}

println "Hi, I'm ${aWeirdName()}"
println (['A', 'B', 'C'].collect {
  "The letter $it"
})
```

```
jkelly@ec-devcave9 groovy/nice_stuff » groovy ImplicitReturn.groovy
Hi, I'm Jon Kelly
[The letter A, The letter B, The letter C]
```

Groovy Truth

- Groovy borrows some of the concepts of “Truthy” and “Falsey” from languages like JavaScript
- Empty lists and maps, iterators with no more elements, empty strings and nulls are coerced to false
- Regex Matchers with at least one match, non-zero numbers and non-null objects are coerced to true

```
assert ![]
assert ![:]
assert ![].iterator()
assert !''
assert !null
assert !0

assert "Hi"
assert ['one': 1]
assert ['A']
assert 42
assert ("Hello World" =~ /World/)
```

Default method params

- Groovy can handle default parameters when declaring a method
- This basically does traditional Java “telescoping” behind the scenes

```groovy
def addToInt(i, amt=1) {
   i + amt
}

assert addToInt(1) == 2
assert addToInt(1, 3) == 4

//Groovy makes these for you
def addToInt(i) {
   addToInt(1, 1)
}

def addToInt(i, amt) {
   i + amt
}
```

Multiple Assignment

- ☐ Multiple variables can be assigned at once.
- ☐ Left side: variable names in parentheses
- ☐ Right side: array of values

```groovy
def (a,b,c) = [10,20,'foo']
assert a == 10 && b == 20 && c == 'foo'

def nums = [1,3,5]
def d,e,f
(d,e,f) = nums
assert d == 1 && e == 3 && f == 5

def (_, m, y) = "18th June 2009".split()
assert "In $m of $y" == "In June of 2009"

def (g,h,i) = [1,2]
assert g == 1 && h == 2 && i == null

def (j,k) = [1,2,3]
assert j == 1 && k == 2
```

Java

```java
String a = "A";
String b = "B";

String tmp = a;
a = b;
b = tmp;
```

Groovy

```groovy
String a = "A"
String b = "B"

(b,a) = [a,b]
```

Ranges

- Range syntax to create lists of sequenced data

```
println 1 .. 5
println 'A' .. 'D'
```

```
jkelly@ec-devcave9 groovy/nice_stuff » groovy Range.groovy
[1, 2, 3, 4, 5]
[A, B, C, D]
```

```groovy
class Chipotle implements Comparable<Chipotle> {
  static meatWeight = [
    'Chicken':1, 'Steak':2, 'Barbacoa':3, 'Carnitas':4
  ]

  private weight
  def meat

  Chipotle(meat) {
    this.meat = meat
    weight = meatWeight[meat]
  }

  int compareTo(Chipotle o2) {
    this.weight.compareTo(o2.weight)
  }

  String toString() {
    meat
  }

  Chipotle next() {
    def m = meatWeight.find {k, v -> v == weight + 1 }
    m ? new Chipotle(m.key) : null
  }

  Chipotle previous() {
    def m = meatWeight.find {k, v -> v == weight - 1 }
    m ? new Chipotle(m.key) : null
  }
}

println (new Chipotle("Steak") .. new Chipotle("Carnitas"))
```

```
jkelly@ec-devcave9 groovy/nice_stuff » groovy ChipotleRange.groovy
[Steak, Barbacoa, Carnitas]
```

Operators

- <=> “Spaceship” for comparisons
- “in” as shorthand for collection.contains() (1 in [1,2] == true)
- ?. for safe navigation - No more null pointers! (a?.b?.c returns null even if a or b are null)
- ?: “Elvis” for shortened ternary (a ? a : “null” == a ?: “null”)

Operators - Spread (*.)

- Invoke an action on each element of an aggregate object
- Can be a method call or property access

```
assert ['cat', 'elephant']*.size() == [3,8]
assert ['a': 1, 'b': 2, 'c': 3]*.value == [1,2,3]
```

Operators - as

- Casts one type to another
- Can be smarter than straight Java type casting

```groovy
def myInt = 4
def bigDec = myInt as BigDecimal

assert bigDec.class == BigDecimal.class
assert bigDec == 4.0

myInt = "4.2"
bigDec = myInt as BigDecimal

assert bigDec == 4.2
```

Regex operators

- =~ : Find with a regex (creates a Matcher)
- ==~ : Get a match with a regex (returns true or false)

```
assert "cheesecheese" =~ /cheese/

def matcher = "cheesecheese" =~ /cheese/
def ch = matcher.replaceFirst("nice")
assert ch == "nicecheese"

assert "2009" ==~ /\d+/
```

Even Groovier

Advanced Cool Guy Stuff

Metaprogramming

- Groovy gives you access to the internals of a class
- This allows you to access, add, and modify properties directly
- Thanks to closures, we can also add methods to classes dynamically

The meta class

- Accessed by .metaClass, on either a Class or instance
- Can inspect via .properties and .methods
- Also has .hasProperty and .respondsTo methods to test whether an object has a certain property or method

```
class Impenetrable {
  private static random = new Random()
  static generate() {
    def result
    switch (random.nextInt(15)) {
      case 0..4:
        result = new Opaque()
        break
      case 5..9:
        result = new Void()
        break
      case 10..14:
        result = new Obscure()
        break
    }
    result
  }
}
```

```
class Opaque extends Impenetrable {
  String what = "A brick wall, bud"

  def whatAmI() {
    what
  }
}
```

```
class Void extends Impenetrable {
  String me = "Nothingness"

  def whatAmI() {
    me
  }
}
```

```
class Obscure extends Impenetrable {
  String me = "Nunya Bizness"

  def whoAmI() {
    me
  }
}
```

Inspecting random Impenetrables

```
def o
for (i in 1..10) {
  o = Impenetrable.generate()
  println "--== Subject $i ==--"
  if (o.metaClass.hasProperty(o, "me")) {
    println "  Looks like a person..."
  } else {
    println "  Doesn't look like a person..."
  }

  if (o.metaClass.respondsTo(o, "whoAmI")) {
    println "  Who are you? ... ${o.whoAmI()}"
  } else if (o.metaClass.respondsTo(o, "whatAmI")) {
    println "  Not a person... must be ${o.whatAmI()}"
  }
}
```

Result

```
jkelly@ec-devcave9 groovy/metaprogramming » groovy Inspect.groovy
--== Subject 1 ==--
  Doesn't look like a person...
  Not a person... must be A brick wall, bud
--== Subject 2 ==--
  Looks like a person...
  Not a person... must be Nothingness
--== Subject 3 ==--
  Doesn't look like a person...
  Not a person... must be A brick wall, bud
--== Subject 4 ==--
  Looks like a person...
  Who are you? ... Nunya Bizness
--== Subject 5 ==--
  Doesn't look like a person...
  Not a person... must be A brick wall, bud
--== Subject 6 ==--
  Doesn't look like a person...
  Not a person... must be A brick wall, bud
--== Subject 7 ==--
  Doesn't look like a person...
  Not a person... must be A brick wall, bud
--== Subject 8 ==--
  Doesn't look like a person...
  Not a person... must be A brick wall, bud
--== Subject 9 ==--
  Looks like a person...
  Not a person... must be Nothingness
--== Subject 10 ==--
  Doesn't look like a person...
  Not a person... must be A brick wall, bud
```

Another method - Case

```
def o
for (i in 1..10) {
  o = Impenetrable.generate()
  println "--== Subject $i ==--"
  switch (o) {
    case Opaque:
    case Void:
      println "Must be ${o.whatAmI()}"
      break
    case Obscure:
      println "Must be ${o.whoAmI()}"
      break
  }
}
```

More case stuff

```
def o
for (i in 1..10) {
  o = Impenetrable.generate()
  println "--== Subject $i ==--"
  switch (o) {
    case Opaque:
    case Void:
      switch (o.whatAmI()) {
        case ~ /.*brick.*/:
          println "Must be Mr. Brick Wall"
          break
        case ~ /Nothing.*/:
          println "Must be Mr. Void"
          break
        default:
          println "Must be ${o.whatAmI()}"
      }
      break
    case Obscure:
      println "Must be ${o.whoAmI()}"
      break
  }
}
```

Method addition

- [ ] You can dynamically add a method to a class, and it's not even hard

But first - delegates

- Each closure has a few variables
- “this” - The instance of the class the closure was defined in
- “owner” - Same as this unless the closure was defined inside another closure
- “delegate” - Same as owner by default, but it can be changed programmatically

Not Even Hard

```groovy
String.metaClass.swapCase = { ->
  def sb = new StringBuffer()
  def ch
  delegate.each {
    ch = it as char
    sb << (Character.isUpperCase(ch) ? Character.toLowerCase(ch) :
           Character.toUpperCase(ch))
  }
  sb.toString()
}

println "TuNe-YaRdS".swapCase()
```

```
jkelly@ec-devcave9 groovy/metaprogramming » groovy SwapCase.groovy
tUnE-yArDs
```

```groovy
String.metaClass.upperSome = { chars ->
  def sb = new StringBuffer()
  def ch
  delegate.each {
    ch = it as char
    sb << (it in chars ? Character.toUpperCase(ch) : ch)
  }
  sb.toString()
}

println "bohemian rhapsody".upperSome(['b', 'r'])
```

```
jkelly@ec-devcave9 groovy/metaprogramming » groovy UpperSome.groovy
Bohemian Rhapsody
```

```groovy
class FooFinder {
  def MASTER = ["foo"]

  def findIt(list, Closure cl) {
    list.findAll(cl)
  }
}

class BarOrBrahFinder {
  def MASTER = ["bar", "brah"]

  def findIt(list, Closure cl) {
    list.findAll(cl)
  }
}

def list = ['foo', 'baz', 'bar', 'brah', 'breh', 'bluh']

def contains = {
  MASTER.contains(it)
}

println new FooFinder().findIt(list, contains)
println new BarOrBrahFinder().findIt(list, contains)
```

```
jkelly@ec-devcave9 groovy/metaprogramming ➜ groovy Finder.groovy
Caught: groovy.lang.MissingPropertyException: No such property: MASTER for class: Finder
```

```groovy
class FooFinder {
  def MASTER = ["foo"]

  def findIt(list, Closure cl) {
    cl.delegate = this
    list.findAll(cl)
  }
}

class BarOrBrahFinder {
  def MASTER = ["bar", "brah"]

  def findIt(list, Closure cl) {
    cl.delegate = this
    list.findAll(cl)
  }
}

def list = ['foo', 'baz', 'bar', 'brah', 'breh', 'bluh']

def contains = {
  MASTER.contains(it)
}

println new FooFinder().findIt(list, contains)
println new BarOrBrahFinder().findIt(list, contains)
```

```
jkelly@ec-devcave9 groovy/metaprogramming » groovy Finder.groovy
[foo]
[bar, brah]
```

Operators as methods

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

Overloading

How to mess with your friend's Groovy project

```groovy
String.metaClass.plus = { String b ->
  def sb = new StringBuffer()
  sb << b
  sb << delegate
  sb.toString()
}

println ("How" + "Now")
```

```
jkelly@ec-devcave9 groovy/metaprogramming » groovy Overload.groovy
NowHow
```

```
println ("WHAT THE" % "whAT tHe")
```

Groovy interprets this as “WHAT THE”.mod(“whAT tHe”), so…

```
String.metaClass.mod = { String b ->
  delegate.equalsIgnoreCase(b)
}

println ("WHAT THE" % "whAT tHe")
```

```
jkelly@ec-devcave9 groovy/metaprogramming » groovy Mod.groovy
true
```

Categories

- [ ] If you don't feel comfortable extending or changing the internals of a class permanently, you have categories
- [ ] Categories allow you to extend classes only within the scope of a "use" block
- [ ] Borrowed from Objective-C

```groovy
class StringCategory {
  static String lower(String string) {
    string.toLowerCase()
  }
  static String mod(String a, String b) {
    a.equalsIgnoreCase(b)
  }
}

use (StringCategory) {
  assert 'test' == 'TeSt'.lower()
  assert 'test' % 'TeSt'
}

try {
  println 'tESt'.lower()
} catch (Exception e) {
  println "Only can use with a category"
}
```

REPL

- Groovy comes equipped with a command-line shell and a graphical console.
- Makes things easier to test on the fly, much like python, node, etc.

Programmatic Shell

- [ ] You can create the GroovyShell within your program.
- [ ] It can parse and run a script from a file, or just a regular string

```groovy
def code = """
  def generation = "dynamic"
  println generation
"""


Script scr = new GroovyShell().parse(code)

scr.run()
```

```
jkelly@ec-devcave9 groovy/dynamic_code »  groovy shell.groovy
dynamic
```

DSL

- [ ] We can combine metaprogramming with dynamic code execution to create a custom DSL
- [ ] Requires a few acrobatics