FUNCTIONAL GROOVY

ANDRES ALMIRAY
CANOO ENGINEERING A.G.
@AALMIRAY

ABOUT THE SPEAKER

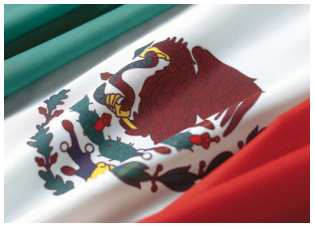
Java developer since the beginning

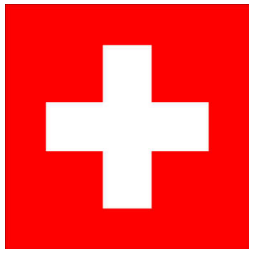
True believer in open source

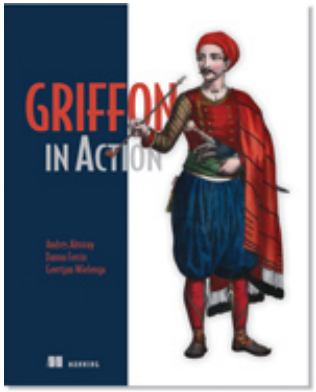
Groovy committer since 2007

Project lead of the Griffon framework

Currently working for canoo








Scala

FUNCTIONAL GROOVY, ARE YOU KIDDING ME?

GROOVY IS NOT HASKELL

RUSSEL WINDER

CLOSURES == FUNCTIONS

Closures are functions (i.e, blocks of code) with an environment containing a binding for all free variables of the function

```
def multiplier = { int base, int factor ->
    base * factor
}

assert 4 == multiplier(2, 2)
assert 4 == multiplier.call(2, 2)
```

CLOSURES == FUNCTIONS

Closures are NOT side effect free by design

```
int var = 0
def multiplier = { int base, int factor ->
    var = 42
    base * factor
}

assert 0 == var
assert 4 == multiplier(2, 2)
assert 4 == multiplier.call(2, 2)
assert 42 == var // !!! ZOMG !!!
```

CLOSURES: PARAMETERS (1)

Parameter types may be omitted if type information is not needed

```
def multiplier = { base, factor ->
    base * factor
}

assert 4 == multiplier(2, 2)
assert 4 == multiplier.call(2, 2)
```

CLOSURES: PARAMETERS (2)

Parameters may have default values

```
def multiplier = { base, factor = 2 ->
    base * factor
}

assert 4 == multiplier(2, 2)
assert 4 == multiplier(2)
assert 6 == multiplier(3)
assert 9 == multiplier(3, 3)
```

NOTE: Default values must be defined from right to left

CLOSURES: DEFAULT PARAMETER

Closures may have a default parameter named it

```
def upper1 = { s -> s.toUpperCase() }
def upper2 = { it.toUpperCase() }

assert 'HELLO' == upper1('hello')
assert 'HELLO' == upper2('hello')

def noArgClosure = { -> 'none' }

assert 'none' == noArgClosure()
```

CLOSURES LEAD TO ...

Partial Evaluation

Composition

Memoization

Tail calls

Iterators

Streams

PARTIAL EVALUATION (1)

Currying creates a new closure with fixed parameters, left to right

```
def m = { x, y -> y / x }
assert 1 == m(1, 1)
assert 2 == m(1, 2)

def xAt5 = m.curry(5)
assert 0.2 == xAt5(1)
```

PARTIAL EVALUATION (2)

Currying may be applied right to left too, even on an arbitrary index

```
def f = { a, b, c ->
    a + b + c
}
def g = f.curry('G')
def h = f.rcurry('H')
def k = f.ncurry(1, 'K')

assert '123' == f('1', '2', '3')
assert 'G12' == g('1', '2')
assert '12H' == h('1', '2')
assert '1K2' == k('1', '2')
```

COMPOSITION (1)

Closures may be composed (left to right) using the >> operator

```
def upper = { it.toUpperCase() }
def doubler = { it * 2 }

def transform = upper >> doubler

assert 'AA' == transform('a')
assert 'AA' == doubler(upper('a'))
```

COMPOSITION (2)

Closures may be composed (right to left) using the << operator

```
def upper = { it.toUpperCase() }
def doubler = { it * 2 }

def transform = upper << doubler

assert 'AA' == transform('a')
assert 'AA' == upper(doubler('a'))
```

MEMOIZATION

Cache computed values for increased performance

```
def fib = null
fib = { p ->
    if (p < 2) BigInteger.ONE
    else fib(p - 1) + fib(p - 2)
}.memoize()

    WITHOUT(memoize)
// Fib(5): 8, time: 0.00 sec
// Fib(20): 10946, time: 0.66 sec
    WITH(memoize)
// Fib(5): 8, time: 0.01 sec
// Fib(20): 10946, time: 0.00 sec
```

TAIL CALLS (1)

Recursive closures may use Tail Calls thanks to trampoline()

```
def fib = null
fib = { n, a = ZERO, b = ONE ->
    if(n == 0) a
    else fib.trampoline n - 1, b, a + b
}

fib(1001)
// Fib(1001): 7033036771142281582183525487718354977018126983635873274260490508715453711819693357974224949456261173348775044924176599108818636326545022364710601205337412127386733911119813937312559876769009190224524532340350
// time: 0.00 sec
```

TAIL CALLS (2)

Apply @TailRecursive on methods

```
@groovyx.transform.TailRecursive
def factorial(n, aggregator = ONE) {
    if (n == ONE) return aggregator
    return factorial(n - ONE, n * aggregator)
}

factorial(5000)

// StackOverflowError without @TailRecursive
```

https://github.com/jlink/tailrec/

ITERATORS (1)

```
def list = [1, 2, 3, 4, 5]

assert 15 == list.inject { e , a -> a += e }
assert 15 == list.sum()
assert [1, 3, 5] == list.findAll { it % 2 }
assert [1, 3, 5] == list.grep { it % 2 }
assert 3 == list.find { it > 2 }
assert list.every { it < 6 }
assert list.any { it % 2 == 0 }
assert '1, 2, 3, 4, 5' == list.join(', ')
```

ITERATORS (2)

```
def list = [1, 2, 3, 4, 5]

assert [1, 2] == list.take(2)
assert [4, 5] == list.drop(3)
assert [4, 5] == list.dropWhile { it <= 3 }
assert 1 == list.head()
assert [2, 3, 4, 5] == list.tail()
assert [6, 8] == list[2..3].collect { it * 2 }
assert [[1, 2, 3], [4, 5]] == list.collate(3)
```

ITERATORS

OBJECTS AS PARTIAL EVALS

Any class may implement the call() method, enabling implicit evaluation

```
class Taxer {
    def factor

    def call(capital) { capital * factor }
}

def taxer = new Taxer(factor: 0.15)
assert 150 == taxer(1000)
taxer.factor = 0.25
assert 250 == taxer(1000)
```

METHODS AS CLOSURES

Any method may be transformed to a Closure using the .& operator

```
class Calculator {
    static square(a) { a * a }
}

def sqr = Calculator.&square
assert 25 == sqr(5)
def list = [1, 2, 3]
assert [1, 4, 9] == list.collect(sqr)
```

STREAMS (1)

Lazy generators. Extension module created by @tim_yates

```
@Grab('com.bloidonia:groovy-stream:0.5.2')
import groovy.stream.Stream

Stream s = Stream.from { 1 }
assert s.take(5).collect() == [1, 1, 1, 1, 1]

s = Stream.from 1..10 filter { it % 2 == 0 }
assert s.collect() == [2, 4, 6, 8, 10]
```

STREAMS (2)

Groovy is Java friendly. Usa any Java library such as functional-java

```
@Grab('org.functionaljava:functionaljava:3.0')
import fj.data.Stream

Stream.metaClass.filter = { Closure c ->
    delegate.filter(c as fj.F) }
Stream.metaClass.asList = {
    delegate.toCollection().asList() }

def evens = Stream.range(1).filter{ it % 2 == 0 }
assert [2, 4, 6, 8, 10, 12] == evens.take(6).asList()
```

IMMUTABILITY

The @Immutable AST transformation makes writing immutable classes trivial

```
@groovy.transform.Immutable
class ImmutablePerson { String name }

person1 = new ImmutablePerson('Duke')
person2 = new ImmutablePerson(name: 'Duke')
assert person1 == person2

shouldFail(ReadOnlyPropertyException) {
    person1.name = 'boom!'
}
```

GPARS

http://gpars.codehaus.org/

- Concurrent collection processing
- Composable asynchronous functions
- Fork/Join abstraction
- Actor programming model
- Dataflow concurrency constructs
- CSP
- Agent - an thread-safe reference to mutable state

PARALLEL COLLECTIONS

Gpars enhances JDK/GDK collections with parallel execution enabled versions

```
GParsPool.withPool {
    def selfPortraits = images.findAllParallel{
        it.contains me}.collectParallel {it.resize()
    }

    //a map-reduce functional style
    def smallestSelfPortrait = images.parallel
        .filter{it.contains me}
    .map{it.resize()}
    .min{it.sizeInMB}
}
```

RESOURCES

- http://pragprog.com/magazines/2013-01/using-memoization-in-groovy
- http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=functional+thinking:
- https://github.com/jlink/tailrec/
- http://timyates.github.com/groovy-stream/
- http://www.jroller.com/vaclav/
- http://gpars.codehaus.org/
- http://www.slideshare.net/arturoherrero/functional-programming-with-groovy

Q & A

THANK YOU!