



# The Groovy Programming Language

Let's Get Groovy!

**Rod Cope and James Strachan**  
<http://groovy.codehaus.org>



[java.sun.com/javaone/sf](http://java.sun.com/javaone/sf)





# Groovy Goal

What You'll Get Out Of This Session

---

Learn what Groovy can do for you and  
how to start using it today!

---



# Agenda

- Background
- What is Groovy?
- Language Details
- Working with Java
- Groovy Markup
- Extras
- Conclusion



# Agenda

- **Background**
- What is Groovy?
- Language Details
- Working with Java
- Groovy Markup
- Extras
- Conclusion

# Java's biggest strengths?

- VM and Binary Compatibility
  - We can build deployment units (class, jar, jnlp, war, ear, rar, car) and run them anywhere
  - We can easily reuse libraries, APIs and tools
- Lots of reusable software & components & tools
- We can innovate at the source code level if we play nice at the binary level

# Components vs. scripting

- We are reusing far more code than we write
- We spend much more time these days gluing components/libraries/tools together than we do writing components
  - We probably spend more time writing tests than we do writing real code :)
  - We're all TDD infected right?
- The web tier is a perfect example of this
  - Render business objects as markup (templating / scripting)
  - Gluing actions with requests and domain models (MVC)
  - Most of the web tier is duct tape with small pockets of domain models (YMMV)
- Scripting is a great way to glue components together

# So why make another agile language?

- Desire for complete binary compatibility with Java
  - No difference at the JVM / bytecode level to Java code
  - No wrappers or separate islands of APIs
    - allow easy mix and match of Groovy & Java code
- Java friendly syntax
  - We've enough to worry about as enterprise developers without unnecessary low level language syntax differences
- Reuse the J2SE / J2EE APIs
  - Build on the Java platform, rather than porting an existing platform inside Java which leads to leaky abstractions
  - No need to learn a new set of APIs such as in Jython / JRuby
  - Help make J2SE and J2EE APIs easier to use
- A scripting language specifically designed for use in the Java Platform by Java developers for Java developers



# Agenda

- Background
- **What is Groovy?**
- Language Details
- Working with Java
- Groovy Markup
- Extras
- Conclusion

# Groovy

- Who/When
  - James Strachan, Bob McWhirter – August 2003
  - Its all Bob's fault! :-)
- What
  - Dynamic, object-oriented scripting language for the JVM
  - Features of Ruby, Python, Dylan and Smalltalk
- How
  - Hand-written compiler and bytecode generator



# Features

- Static & dynamic typing
- Native syntax for lists, maps, arrays, beans etc
- Closures
- Regex
- Operator overloading
- Autoboxing and polymorphism across collection, array, map, bean, iterators etc
- Making the JDK more Groovy with new methods

# Java and Groovy!

```
public class Filter {  
    public static void main( String[] args ) {  
        List list = new java.util.ArrayList();  
        list.add( "Rod" ); list.add( "James" ); list.add( "Chris" );  
        Filter filter = new Filter();  
        List shorts = filter.filterLongerThan( list, 4 )  
        for ( String item : shorts ) { System.out.println( item ); }  
    }  
    public List filterLongerThan( List list, int length ) {  
        List result = new ArrayList();  
        for ( String item : list ) {  
            if ( item.length() <= length ) { result.add( item ); }  
        }  
        return result;  
    }  
}
```

# Groovy!

```
list = ["Rod", "James", "Chris"]  
shorts = list.findAll { it.size() <= 4 }  
shorts.each { println it }
```

-> Rod



# Agenda

- Background
- What is Groovy?
- **Language Details**
- Working with Java
- Groovy Markup
- Extras
- Conclusion

# Optional Things

- Optional Variable Declarations and Types

```
int a = 3;  
b = 2;  
String t = "hello";  
s = 'there';
```

- Optional Semicolons and Parentheses

```
int a = 3; int b = 4;  
println( a );  
c = 2  
print c  
print "hello".indexOf( 'e' )
```

# Strings

- Sample Declarations
  - `h1 = "hellohello"`
  - `h2 = 'hello' * 2` (same as `h2 = 'hellohello'`)
  - `h1 == h2` but not *necessarily* `h1 === h2`
  - `mix1 = "hello 'there' dude"`
  - `mix2 = 'hello "there" dude'`
  - `wrap = "this is  
a string that wraps lines"`
  - `triple = """This string  
doesn't need "quote" escaping 'at all' dude"""`
  - `heredoc = <<<BLAH  
This is some text that  
doesn't need ""any"" kind of quote 'escaping'  
BLAH`

# GStrings

- Sample Declarations

- `name = "Rod"`

- `message = "hello ${name}"`

- `longMessage = """Hi ${name}!`

- You may have already won `${calc(name)}` million dollars!"""

- Evaluated at time of first use

- Doesn't expand within outermost single quotes

- `println "${name}"`    `-> Rod`

- `println '${name}'`    `-> ${name}`

- Optional Method "return"

- `calc(name) { name.size() }`

# Data Structures

- Lists:
  - `list = [1, 3, 'apple', "dog"]`
  - `list.size() == 3`
  - `list + [4, 'cat'] == [1, 3, 'apple', 'dog', 4, 'cat']`
  - `list - ['apple'] == [1, 3, 'dog']`
  - **Empty list:** `[]`
- Maps:
  - `map = ['rod':33, 'james':35]`
  - `map.size() == 2`
  - **Empty map:** `[:]`

# Indexing 1

- Strings:
  - `str = 'testing'`
  - `str[2] == 's'`
  - `str[-1] == 'g'`
  - `str[1..2] == 'es'`
  - `str[1...3] == 'es'`
  - `str[3..2] == 'ts'`
  - `str[4,2] == 'is'`
- Lists and Arrays:
  - `list = [1, 3, 'apple', "dog", 'bug', "tree"]`
  - `list[2] == 'apple'`
  - `list[-1] == 'dog'`
  - `list[1..2] == [3, 'apple']`
  - `list[1...3] == [3, 'apple']`
  - `list[3..2] == ['dog', 'apple']`
  - `list[4,2] == ['bug', 'apple']`

# Indexing 2

- Maps:

```
- map = ['rod' : 33, 'james' : 35]
- println map['rod']
  -> 33
- println map.rod
  -> 33
- map.joe = 28
  -> ['rod':33, 'james':35, 'joe':28]
- println map.subMap(['rod', 'joe'])
  -> ['rod':33, 'joe':28]
```

# Basic Iteration and Looping

- Samples
  - Simple loops: 

```
for ( index in 1..3 )  
  { println index }
```
  - Strings: 

```
for ( ch in 'dog' )  
  { println ch }
```
  - Lists and Arrays: 

```
for ( item in [1,2,'dog'] )  
  { println item }
```
  - Maps: 

```
for (entry in ['rod':33,'james':35])  
  {println "${entry.key} = ${entry.value}" }
```
- Support for Java while, do, if, else, etc.
- No standard Java 'for' implemented yet

# Closures

```
// sum things
sum = 0
[1, 2, 3].each { |item| sum += item }
println "The total is ${sum}"

// make a text list of customer names
text = customers.collect { |item|
    return item.name }.join(", ")

// make a closure then call it later on
C = { |name| println "Hey ${name}" }
C("James")
```

# Iteration with Closures

- Basic Iteration
  - Simple loops: `5.times { print it }`  
-> 01234
  - Strings: `'dog'.each { |c| println c }`  
-> d  
o  
g
  - Lists and Arrays: `[1,2,'dog'].each { print it }`  
-> 12dog
  - Maps: `['rod':33, 'james':35].each {  
print "${it.key}=${it.value} " }`  
-> james=35 rod=33

# Regular Expressions (Regex)

- Patterns

- `pattern = ~"na.*"`

- Matchers

- `matcher = ("name" =~ pattern)`

- `println matcher.matches() -> true`

- Regex Matches

- `println( "name" =~ "na.*" ) -> true`

- Indexing Matchers

- `m = "a name is just a game" =~ ".ame"`

- `println m[1] -> "game"`

- `m = "a name is just a game" =~  
".*name (.*) just (.*) game.*"`

- `println( "Regex ${m[1]} ${m[2]} cool thing!" )  
-> "Regex is a cool thing!"`

# switch

- `switch (x)`  
{  
    `case 7:`                   `println "7";`           `break;`  
    `case 2.3:`                  `println "2.3";`       `break;`  
    `case "dog":`                `println "dog";`       `break;`  
    `case [1, 2, 'alf']:`       `println "in list";`   `break;`  
    `case (3..6):`               `println "in range";` `break;`  
    `case Integer:`             `println "Integer";`   `break;`  
    `case ~"ca.*":`             `println "regex";`     `break;`  
    `case new MyCase():`        `println "MyCase";`    `break;`  
}
- `class MyCase { isCase(thing) {return thing == 'me'} }`

# Groovy JDK

- Adds methods missing from the JDK
- String
  - contains(), count(), execute(), padLeft(), center(), padRight(), reverse(), tokenize(), each(), etc.
- Collection
  - count(), collect(), join(), each(), reverseEach(), find/All(), min(), max(), inject(), sort(), etc.
- File
  - eachFile(), eachLine(), withPrintWriter(), write(), getText(), etc.
- Lots there and growing all the time
- You can add methods programmatically

# Basic Sorting

- `list = [ 'dog', 'bird', 'chick' ]`
- `println( list.sort() )`  
-> `[bird, chick, dog]`
- `println( list.sort { it.size() }.reverse() )`  
-> `[chick, bird, dog]`

# Sorting JavaBeans

```
class Person { name; age }
list = [ new Person( name:'Rod', age:33 ),
         new Person( name:'James', age:35 ) ]
list.sort { | person | person.age }
list.sort { [ it.name, it.age ] }
list.sort { | a, b | a.name <=> b.name }
println( list.sort{it.name}.name )
    -> [James, Rod]
println(list.sort{it.name}.name.join(':'))
    -> "James:Rod"
```

# GPath

```
class Person { name; age }  
list = [ new Person( name:'Rod', age:33 ),  
         new Person( name:'James', age:35 ) ]  
println( list.find {it.age > 25}.name )  
    -> [Rod]  
println( list.findAll {it.age > 25}.name )  
    -> [Rod, James]  
println( list.any{ it.name.size() > 4 } )  
    -> true
```

# Real World GPath

```
for ( order in customers.findAll {
    it.location.code == "CO" }.orders )
{
    println( "${order.id} is ${order.value}" )
}
```

# Writing grep in Groovy

- `// grep.groovy`  
`new java.io.File(args[0]).eachLine {`  
    `if (it =~ ".*${args[1]}.*" ) {`  
        `println it`  
    `}`  
`}`
- `groovy grep.groovy myfile.text dog`

# Groovy! (Revisited)

```
list = ["Rod", "James", "Chris"]  
shorts = list.findAll { it.size() <= 4 }  
shorts.each { println it }
```

-> Rod



# Agenda

- Background
- What is Groovy?
- Language Details
- **Working with Java**
- Groovy Markup
- Extras
- Conclusion



# Using Java in Groovy Code

- Just use it!
- Inherit from or use any Java code as usual

# Using Groovy in Java Code

(A)

- BSF-compliant

- Ant: `<script language="groovy">  
println 'hi' * 2</script>`

- Scripting In-Place

```
Binding binding = new Binding();  
binding.setVariable( "foo", new Integer(2) );  
GroovyShell shell = new GroovyShell( binding );  
Object value = shell.evaluate(  
    "println 'Hello World!'; x = 123; return foo * 10");  
assert value.equals( new Integer(20) );  
assert binding.getVariable("x").equals(  
    new Integer(123) );
```

# Using Groovy in Java Code (B1)

- ```
class GAdder
{
    public int add( int a, int b )
    {
        return a + b
    }
}
```

# Using Groovy in Java Code

## (B2)

- Using Reflection

```
parent = Thread.currentThread().getContextClassLoader();
loader = new GroovyClassLoader( parent );
groovyClass = loader.parseClass(
    new File("GAdder.groovy"));

groovyObject = (GroovyObject) groovyClass.newInstance();
Object[] args = { new Integer( 1 ), new Integer( 2 ) };
Object answer = groovyObject.invokeMethod("add", args);

assertEquals( new Integer( 3 ), answer );
```

# Using Groovy in Java Code (C1)

- ```
public interface Adder
{
    public int add( int a, int b );
}
```
- ```
class GAdder implements Adder
{
    public int add( int a, int b )
    {
        return a + b
    }
}
```

# Using Groovy in Java Code

(C2)

- Using Interfaces

```
parent = Thread.currentThread().getContextClassLoader();
loader = new GroovyClassLoader( parent );
groovyClass = loader.parseClass(
    new File("GAdder.groovy"));

adder = (Adder) groovyClass.newInstance();
int answer = adder.add( 1, 2 );

assertEquals( new Integer( 3 ), answer );
```



# Agenda

- Background
- What is Groovy?
- Language Details
- Working with Java
- **Groovy Markup**
- Extras
- Conclusion

# XML: Generation

```
data = [ 'Rod':  [ 'Misha':8, 'Bowie':2],
         'Eric': [ 'Poe':4, 'Doc':3] ]

xml = new groovy.xml.MarkupBuilder()

people = xml.people() {
    for ( entry in data ) {
        person( name: entry.key ) {
            for ( dog in entry.value ) {
                pet( name:dog.key, age:dog.value )
            }
        }
    }
}
```

# XML: Generation Results

```
<people>
  <person name='Rod'>
    <pet name='Bowie' age='2' />
    <pet name='Misha' age='8' />
  </person>
  <person name='Eric'>
    <pet name='Poe' age='4' />
    <pet name='Doc' age='3' />
  </person>
</people>
```

# XML: Parsing & Navigation

```
people = new groovy.util.XmlParser().parseText( xml )
println people.person.pet['@name']
    -> ['Misha', 'Bowie', 'Poe', 'Doc']

println people.person.pet.findAll {
    Integer.parseInt( it['@age'] ) > 3 }['@name']
    -> [Misha, Poe]

println people.person.findAll {
    it.pet['@name'].every { it.size() > 3 } }['@name'][0]
    -> "Rod"
```

# Groovy Ant (Gravy)

```
class Build {
    ant = new groovy.util.AntBuilder()
    targets = ['clean', 'compile']

    static void main(args) {
        b = new Build()
        if ( args.size() > 0 ) { b.targets = args }
        b.run()
    }
    void run() {
        for ( target in targets ) {
            invokeMethod( target.toString(), null )
        }
    }
    clean() { ant.rmdir( dir:'target' ) }
    compile() {
        ant.mkdir( dir:'target/classes' )
        ant.compile( srcdir:'src/main/java',
                    destdir:'target/classes' ) {
            fileset { includes( name:'**/*.java' ) }
        }
    }
}
```

# Groovy Swing

```
swing = new groovy.swing.SwingBuilder()
frame = swing.frame(title: 'This is a Frame',
                    location:[100,100],size:[800,400]) {
    menuBar {
        menu(text: 'File') {
            menuItem() {
                action(name:'New', closure:{ println("New") })
            }
        }
    }
    panel(layout: new BorderLayout()) {
        label(text: 'Name', constraints: BorderLayout.WEST,
            tooltipText: 'This is the name field')
        textField(text: 'Rod',
            constraints: BorderLayout.CENTER)

        button(text: 'Click me!',
            constraints: BorderLayout.SOUTH,
            actionPerformed:{println("Button clicked!")})
    }
}
```

# Groovy Swing - Tables

```
panel(layout:new BorderLayout()) {
  scrollPane(constraints:BorderLayout.CENTER) {
    table() {
      model = [ ['name':'Rod', 'age':33],
                ['name':'James', 'age':35] ]

      tableModel(list: model) {
        closureColumn( header: 'Name',
                       read: { | row | row.name } )

        closureColumn( header: 'Age',
                       read: { | row | row.age } )
      }
    }
  }
}
```

# Groovy SQL

```
sql = new groovy.sql.Sql( dataSource )  
sql.execute( "create table person  
             ( name varchar, age integer)" )  
  
people = sql.dataSet( "person" )  
people.add( name: "Rod", age: 33 )  
people.add( name: "James", age: 35 )  
sql.eachRow( "select * from person" ) { | p |  
    println "${p.name} is ${p.age} years old"  
}  
  
-> Rod is 33 years old  
-> James is 35 years old
```



# Agenda

- Background
- What is Groovy?
- Language Details
- Working with Java
- Groovy Markup
- **Extras**
- Conclusion

# Groovy Shell

- Easy way to write scripts and test code

```
osprompt> groovysh
```

```
1> ['dog', 'cat', 'bird'].each { | animal |  
    println "I like ${animal}s" }
```

```
2> go
```

```
I like dogs
```

```
I like cats
```

```
I like birds
```

# Groovy Shell Tips

- Tip: Can't remember method names?

```
1> println java.io.File.methods.name.sort()
```

```
2> go
```

```
[canRead, canWrite, compareTo, compareTo,  
createNewFile, createTempFile,  
createTempFile, delete, deleteOnExit, ...]
```

- Tip: Need a directory listing?

```
println "cmd /c dir".execute().text
```

```
println "ls -la".execute().text
```



# Agenda

- Background
- What is Groovy?
- Language Details
- Working with Java
- Groovy Markup
- Extras
- **Conclusion**

# Trouble in Paradise

- Weak and Missing Features
  - No support for anonymous inner classes
  - Tool support (Eclipse, IntelliJ, etc.)
- Debugging/Scripting Hell
  - Immature parser: hard to find "real" bugs
  - Lots of rope: can hang yourself with dynamic code
- Language Instability
  - Syntactic sugar: nice to have, but may change
  - Java camp vs. non-Java camp: competing directions



# Groovy JSR

- JSR-241: The Groovy Programming Language
  - Standardized Language Specification
  - Open Source Reference Implementation
- Ongoing Debates
  - Optional Semicolons
  - Optional Parentheses
  - Properties/JavaBean Spec Compliance
  - Closure Variable Scoping
  - Closure Syntax Sugar/Whitespace

# Conclusion

- Status
  - 1.0-beta-5, AKA RC1-snapshot
- Development Time
  - Half that of Java (except for debugging hell factor)
- Performance
  - 20-90% of Java depending on usage
  - Very little tuning so far - waiting on JSR
- Recommendations
  - Ready for small, non-critical projects and scripting
  - Try it! Very easy to learn and lots of fun!



# For More Information

- Groovy Home Page
  - <http://groovy.codehaus.org>
- GDK Javadoc
  - <http://groovy.codehaus.org/groovy-jdk.html>
- JSR-241: Groovy Language Specification
  - <http://www.jcp.org/en/jsr/detail?id=241>

# Q&A

Rod Cope  
CTO of OpenLogic, Inc.

James Strachan  
Partner of Core Developers Network





# The Groovy Programming Language

Let's Get Groovy!

**Rod Cope and James Strachan**  
<http://groovy.codehaus.org>



[java.sun.com/javaone/sf](http://java.sun.com/javaone/sf)

