

# Introduction to Groovy

Drew Wills

Lennard Fuller

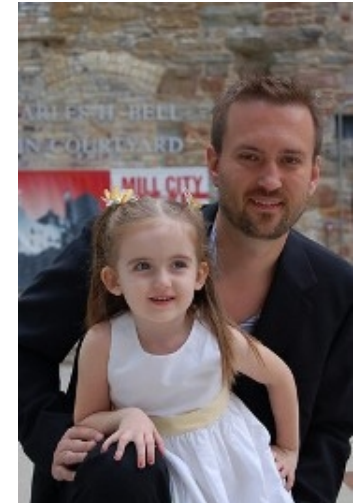
Jasig Spring Conference San Diego, March 7, 2010

© Copyright Unicon, Inc., 2006. This work is the intellectual property of Unicon, Inc. Permission is granted for this material to be shared for non-commercial, educational purposes, provided that this copyright statement appears on the reproduced materials and notice is given that the copying is by permission of Unicon, Inc. To disseminate otherwise or to republish requires written permission from Unicon, Inc.

# Facilitator: Drew Wills



- Joined Unicon November 2000
- JA-SIG Committer since 2004
- Contributed to several uPortal implementations:
  - California Polytechnic State University ("*Cal Poly*")
  - California State University, Chico
  - University of Colorado System
  - Yale University
  - Johns Hopkins University
  - University of Illinois
  - Lynn University
  - myPearson Portal
- Launched Cernunnos February 2007



# Presenter: Lennard Fuller

- Unicon Software Architect
  - Joined Unicon January 2001
- JA-SIG Committer since 2006
- Contributed to and led :
  - Cisco Networking Academy – Virtuoso
  - Pearson Higher Education Portal
    - Largest uPortal deployment serving 8+ million users
  - Pearson HE Communities
    - JCR Application released 2008
  - Pearson Citation Portlet



# Participants

- Please share a bit about yourself:
  - Your name & institution
  - Do you run Jasig software? Which projects? What versions?
  - Have you used Groovy before? How much?
  - What do you hope to learn here?



1. What is Groovy?
2. Groovy Features Overview
3. Section Three
4. Section Four
5. Section Five
6. Section Six

# What is Groovy?

Platform Basics & Project Status

# What is Groovy?



*"Groovy is a powerful high level language for the Java platform which compiles down to Java bytecode.*

*Think of it as a Ruby or Python like language that is tightly integrated with the Java platform - allowing you the same powerful and concise coding syntax as Ruby or Python but allowing you to stay on the JVM and protect your investment in J2SE, J2EE and all the plethora of great useful Java code out there."*

**Groovy FAQ (<http://groovy.codehaus.org/faq.html>)**

# Groovy *IS*

- An agile, dynamic language for the Java Platform
- Easy to learn for Java developers
- Able to mix seamlessly with code written in the Java language



# Groovy *IS NOT*

- Something *other than* Java
- Slow (though it's slower than Java, and 1.0 was much slower)
- Compiled at runtime (unless you want it to be)

# High-Level Feature Summary

- Advantages of Groovy fall mostly into these categories:
  - Better defaults
  - Loads of *syntactic sugar*
  - Closures
  - JDK library enhancements
  - Metaprogramming (a.k.a. *monkey patching*)

# Groovy Status

- Version 1.7.1 released February 19, 2010
- JSR-241: The Groovy Programming Language
  - *Approved March 16, 2004*
- SpringSource Acquires G2One Inc.
  - *November 11, 2008*
- 40<sup>th</sup> most popular programming language as of February, 2010



# A word on performance

Groovy is SLOWER than java

- Not a focus for the groovy community.
  - Primarily focused on ease of dev
- Interpretation on top of the jvm
- Historically true, still true... but to a lesser extent.

**Remain calm during the next slide**

# Occasionally... it can be a much slower

A Quicksort comparison using java 1.6.17,  
Groovy 1.7

n	100,000	1,000,000
java	12	95
Groovy	3750	41700

With regards to performance, the question at hand is 'Is it fast enough?'

# Soo... when should I use groovy?

- Great for command line scripting,
  - works in all envs
- Excellent for unit testing
- Quick proof of concepts
- Applications which have a small user base.
  - Remember, few applications have a user base of millions...

# New in Groovy 1.7

- Anonymous inner/nested classes
- Annotations
- SQL
- Groovy Console
- AST Viewer/Builder
- Full rewrite of the GroovyScriptEngine
- And much much more!

# Getting Started with Groovy

Installation & using Groovy



# Setting up your java env

If you already have java 1.4 or greater installed, skip to “Setting up groovy env”.

- Get the latest Java distribution from <http://java.sun.com>
- Run installer
- Set the JAVA\_HOME environment variables. On Windows, follow these steps:
  - Open the System control panel
  - Click the Advanced tab
  - Click the Environment Variables button
  - Add a new System variable with the name JAVA\_HOME and the value of the directory java was installed in (i..e. [C:\Program Files\java\jdk1.6.0\\_12](#))
  - Optionally add %JAVA\_HOME%\bin to your system path

# Setting up your groovy env

- Unzip groovy-binary-1.7.1.zip from the thumb drive to a logical place on your hard drive. For example C:\dev\groovy-1.7.1
  - You can not have spaces in the path where groovy is installed
- Set the GROOVY\_HOME environment variables. On Windows, follow these steps:
  - Open the System control panel
  - Click the Advanced tab
  - Click the Environment Variables button
  - Add a new System variable with the name GROOVY\_HOME and the value of the directory java was installed in (i.e. C:\dev\groovy-1.7.1 )
  - Optionally add %GROOVY\_HOME%\bin to your system path
  - Try running the groovyConsole.bat by double clicking on it in your GROOVY\_HOME\bin directory

# Running groovy scripts

- **Shell scripts in the groovy home bin/ 'groovy' or 'groovy.bat'**
- **Way to run**
  - **.groovy scripts**
  - **Classes with main methods**
  - **Classes extending GroovyTestCase**
  - **Some classes implementing Runnable**
    - **With Constructor String[] or no args**

```
groovy foo/MyScript.groovy [arguments]
```

# Groovy Shell aka groovysh

- **Command line application**
- **Quick way to**
  - **Evaluate groovy expressions**
  - **Define classes**
  - **Run simple experiments**
- **Simply run groovysh or groovysh.bat inside your groovy bin/**

# groovysh

## Let's test it out!

```
sparhk@bob-laptop:~$ groovysh --help
```

```
usage: groovysh [options] [...]
```

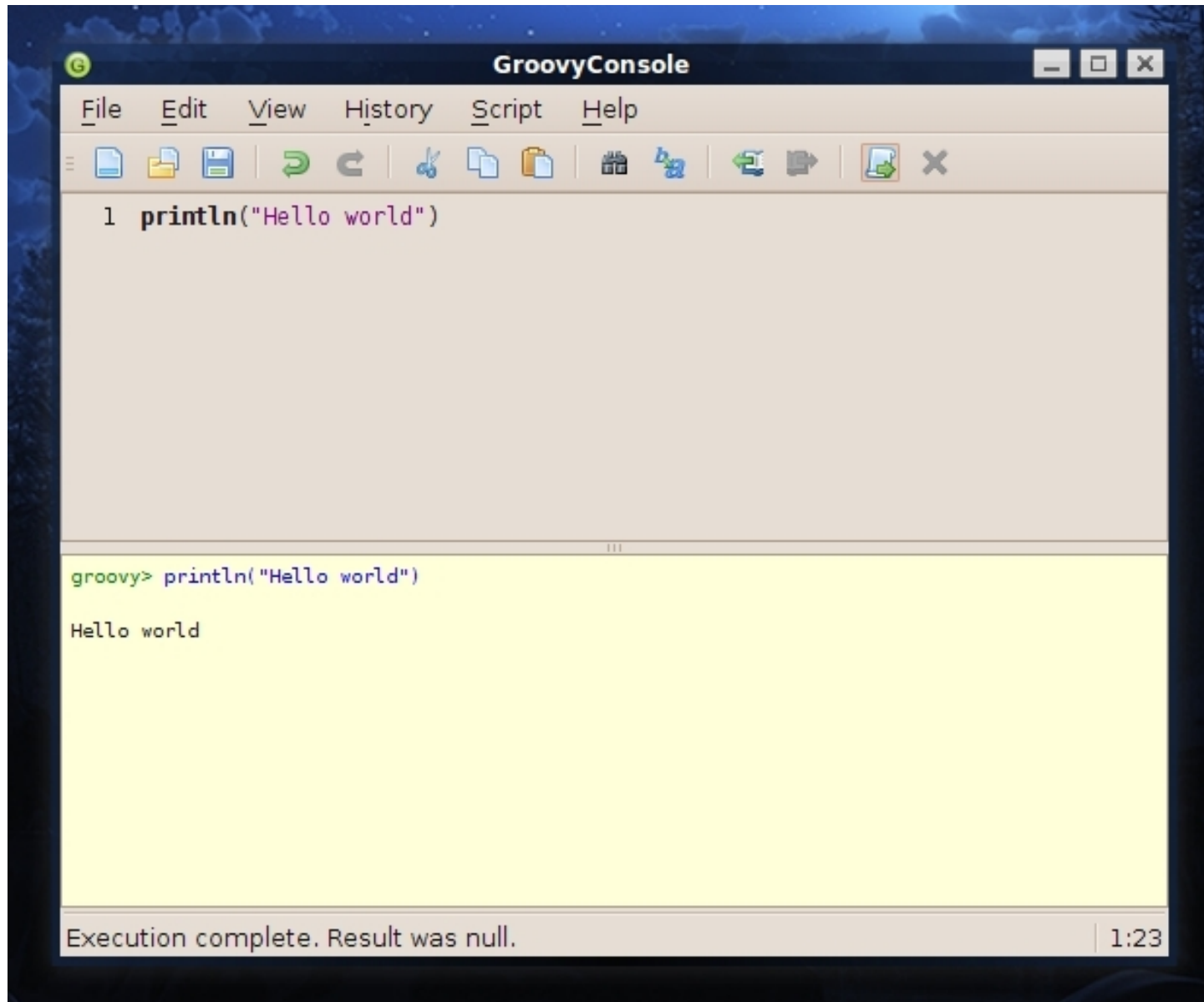
```
  -C, --color[=FLAG]          Enable or disable use of ANSI
colors
  -D, --define=NAME=VALUE     Define a system property
  -T, --terminal=TYPE         Specify the terminal TYPE to use
  -V, --version                Display the version
  -d, --debug                  Enable debug output
  -h, --help                   Display this help message
  -q, --quiet                  Suppress superfluous output
  -v, --verbose                Enable verbose output
```

# groovyConsole

- Swing interactive console
- Type in commands and execute
- History available
  - undo/redo
  - Move forwards and backwards through

```
GROOVY_HOME\bin\groovyConsole.bat
```

# GroovyConsole cont



# Groovy Features Overview

Language & Platform Features



# Language Features

- All of Java, *except...*
  - `==` always means `equals()`
  - `in` is a keyword
  - You can't write `int[] a = {1, 2, 3}` to declare in array (but there's something similar)
  - You can't use the JDK5 for-loop syntax to iterate over a collection, e.g. `for (String s : list)`

# Language Features (cont.)

- All of Java, *although...*
  - These packages & classes are imported by default:
    - **java.io.\***
    - **java.lang.\***
    - **java.math.BigDecimal**
    - **java.math.BigInteger**
    - **java.net.\***
    - **java.util.\***
    - **groovy.lang.\***
    - **groovy.util.\***

# Language Features (cont.)

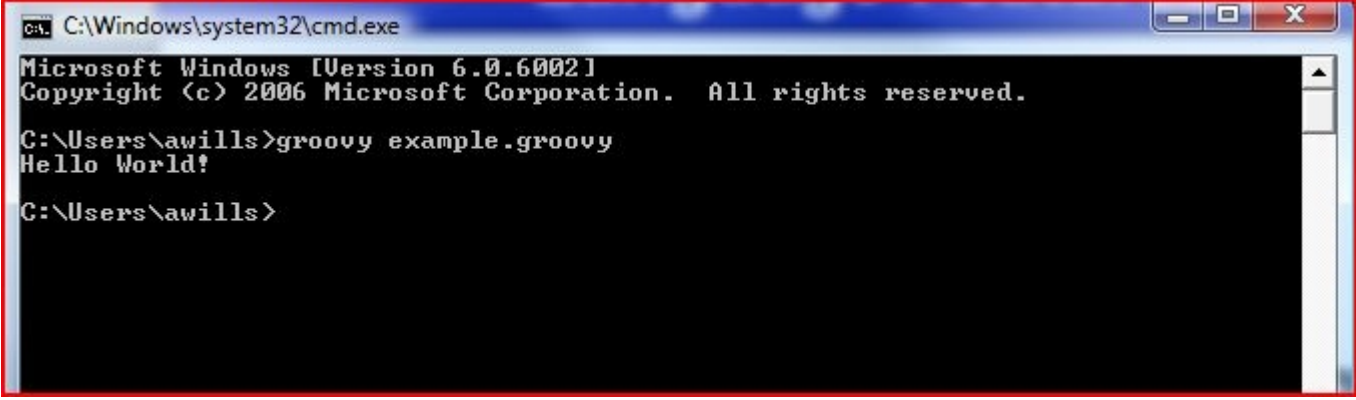
- All of Java, *although...*
  - Semicolons are optional
  - The **return** keyword is optional
  - You may use **this** inside static methods (which refers to the class)
  - Classes & methods are **public** by default
  - The **throws** clause in method signatures is not checked by the Groovy compiler (no checked exceptions in Groovy)
  - You will not get compile-time errors for using undefined members or passing arguments of the wrong type

# Language Features (cont.)

- **print** and **println** statements

```
1 println "Hello World!";  
2 |
```

- This is a complete Groovy program
- You can run it like this...



```
C:\Windows\system32\cmd.exe  
Microsoft Windows [Version 6.0.6002]  
Copyright (c) 2006 Microsoft Corporation. All rights reserved.  
  
C:\Users\awills>groovy example.groovy  
Hello World!  
  
C:\Users\awills>
```

# Language Features (cont.)

- Properties

```
1 class Foo {
2     String bar = "monkey";
3 }
4
5 Foo f = new Foo();
6 f.setBar('baboon');
7 println f.getBar(); // prints "baboon"
8 |
```

- The `def` keyword

```
1 def foo;
2
3 foo = 'One';
4 println foo + 1; // prints "One1"
5
6 foo = 1;
7 println foo + 1; // prints "2"
8 |
```

# Language Features (cont.)

- Safe Navigation Operator ?.

```
1 class Foo {
2     String bar = null;
3 }
4 Foo f = new Foo();
5
6 println f.getBar()?.length(); // prints "null"
7
8 f.setBar('We can invoke length() now');
9 println f.getBar()?.length(); // prints "26"
10 |
```

- Elvis Operator ?:

```
1 def name = null;
2
3 def author = name ?: "Anonymous";
4 println "The source is: " + author;
5 // prints "The source is: Anonymous"
6
7 name = "Drew Wills";
8 author = name ?: "Anonymous";
9 println "The source is: " + author;
10 // prints "The source is: Drew Wills"
11 |
```

# Language Features (cont.)

- String & GString
  - Declare a standard String with single-quote (')
  - Use double-quotes to declare a GString
  - GStrings may contain arbitrary Groovy expressions inside `${...}` delimiters (like EL, Spring, *etc.*)

```
1 class Language {
2     String name;
3 }
4 def grvy = new Language();
5 grvy.setName('Groovy');
6
7 def msg = "Welcome to ${grvy.getName()}!";
8 println msg; // prints "Welcome to Groovy!"
9 |
```

# Language Features (cont.)

- Multi-line Strings

```
1 def title = 'Programming Groovy';
2 def author = 'Venkat Subramaniam';
3
4 def xml = """\
5 <book>
6   <title>${title}</title>
7   <author>${author}</author>
8 </book>
9 """;
10
11 println xml; // prints the following...
12 // <book>
13 //   <title>Programming Groovy</title>
14 //   <author>Venkat Subramaniam</author>
15 // </book>
16
```



# Language Features (cont.)

- *Slashy* strings
  - Do not need backslashes to escape special characters

```
1 def msg = /He told me to "Learn Groovy, it's awesome!"/;  
2 println msg;  
3  
4
```

- This form of string literal is especially handy for...

# Language Features (cont.)

- Native syntax for regex constructs
  - Create a Pattern with the ~ operator

```
1 def pattern = ~/\./;  
2 def tokens = pattern.split('Split.this.String.into.words');  
3 println tokens[4]; // prints "words"  
4  
5
```

- Create a Matcher with the =~ operator

```
1 boolean hasMatch = 'foobar' =~ /foo/;  
2 println hasMatch; // prints true  
3  
4 def m = 'A monkey' =~ /A ([m-o]+)/;  
5 println m[0]; // prints "[A mon, mon]"  
6 println m[0][1]; // prints "mon"  
7  
8
```

# Language Features (cont.)

- Looping with **for..in**

```
1 def name = "Groovy";
2 for (c in name.subSequence(1, 4)) {
3     print c;
4 }
5 // prints "roo"
6 |
```

- Ranges

```
1 for (n in 1..6) {
2     print n;
3 }
4 // prints "123456"
5
6 for (c in 'd'..'k') {
7     print c;
8 }
9 // prints "defghijk"
10
```

# Language Features (cont.)

- **switch** statement
  - Java-compatible, *plus...*
  - Collections
  - Ranges
  - Classes
  - Regex
  - Equality *works with any type!*

# Language Features (cont.)

- **switch statement (cont.)**

```
1  def foo = 7;
2
3  def msg = null;
4  switch (foo) {
5      case [1, 2, 'foo']:
6          msg = 'Switch supports collections';
7          break;
8      case 2..8:
9          msg = 'Switch supports ranges';
10         break;
11     case List.class:
12         msg = 'Switch supports classes';
13         break;
14     case ~/fo+/:
15         msg = 'Switch supports regex expressions';
16         break;
17     case Integer.MAX_VALUE:
18         msg = 'Switch is Java-compatible';
19     default:
20         msg += ', including default & falthrough';
21         break;
22 }
23
24 println msg; // prints "Switch supports ranges"
25 |
```

# Language Features (cont.)

- Native syntax for Lists & Maps

```
1 def list = ['Drew Wills', 'Lennard Fuller'];
2
3 println 'Your instructors are:';
4 for (name in list) {
5     println ' - ' + name;
6 }
7 // prints the following...
8 // Your instructors are:
9 //   - Drew Wills
10 //   - Lennard Fuller
11
12 def map = [title:'Introduction to Groovy', date:'2010/03/07'];
13 println "Welcome to ${map['title']} on ${map['date']}";
14 // prints "Welcome to Introduction to Groovy on 2010/03/07"
15 |
```

- Spread operator \*.

```
1 def list = ['foo', 'bar', 'foobar'];
2
3 println 'My Strings are the following sizes: ' + list*.size();
4 // prints "My Strings are the following sizes: [3, 3, 6]"
5 |
```

# Language Features (cont.)

- Closures
  - A closure is a *unit of work* in the form of an object; they are defined at one point and used at a later point
  - Closures may freely refer to any variable visible within the scope in which they were themselves defined
  - The more familiar you get with Groovy, the more you will use closures

# Language Features (cont.)

- Simple closure example

```
1 def aName = 'Drew Wills';
2 def aClosure = { println 'Hello ' + aName };
3
4 aClosure(); // prints "Hello Drew Wills"
5
6
```

- Closures that return a value

```
1 def counter = 0;
2 def nextValue = { return ++counter };
3
4 for (c in 'd'..'g') {
5     println c + '-' + nextValue();
6 }
7 // prints...
8 // d-1
9 // e-2
10 // f-3
11 // g-4
12
13
```



# Language Features (cont.)

- Passing parameters to a closure

```
1 def sum = { a, b ->
2     return a + b;
3 };
4
5 def arg1 = 4, arg2 = 7;
6 println "The sum of ${arg1} and ${arg2} is " + sum(arg1, arg2);
7 // prints "The sum of 4 and 7 is 11"
8
9
```

- Implicit parameter **it**
  - Closures with one argument may omit the parameter declaration

```
1 def echo = {
2     println it;
3 }
4
5 echo('Boo!'); // prints "Boo!"
6
7
```

# Language Features (cont.)

- Closures as method arguments
  - Methods that take a closure as the *last* parameter can be invoked with a special syntax

```
1 void invokeIt(String name, Closure c) {
2     c(name);
3 }
4
5 invokeIt('Drew') {
6     println 'Closure written by ' + it;
7 };
8 // prints "Closure written by Drew"
9
10
```

# Language Features (cont.)

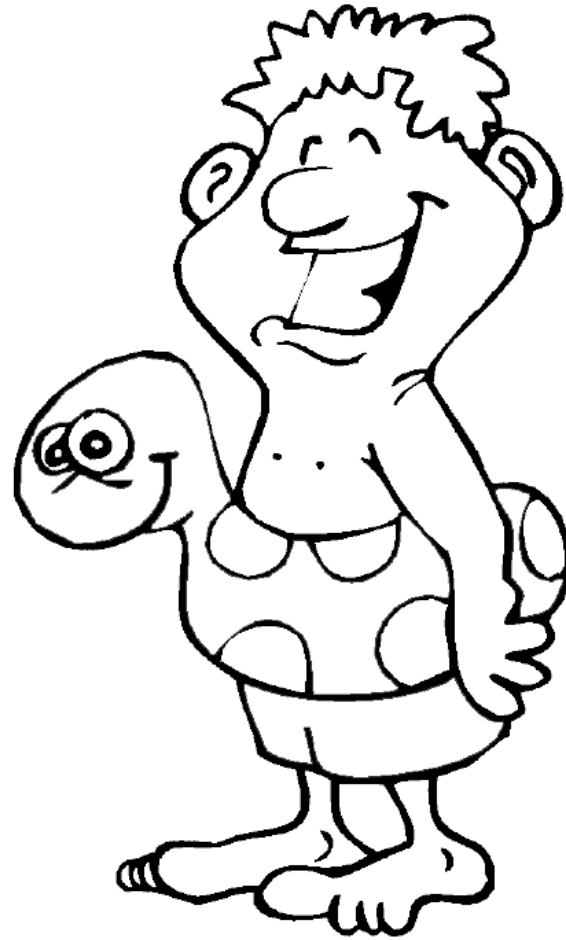
- Keyword **as**
  - Coerce a **Closure** or a **Map** into an interface

```
1 interface Usable {
2     void useIt(String user);
3 }
4
5 void useSomething(Usable u, String name) {
6     u.useIt(name);
7 }
8
9 def mock = { user ->
10     println "Help!  I'm being used by ${user}.";
11 } as Usable;
12
13 useSomething(mock, 'Drew Wills');
14 // prints "Help!  I'm being used by Drew Wills."
15
16
```

# Language Features (cont.)

```
1 import javax.servlet.ServletOutputStream;
2 import javax.servlet.http.*;
3
4 class MySimpleServlet extends HttpServlet {
5     @Override
6     protected void doGet(HttpServletRequest req, HttpServletResponse res) {
7         res.getOutputStream().print('<body>\n').print(' <h1>')
8             .print('Hello ').print(req.getRemoteUser()).print('!!')
9             .print('</h1>\n').println('</body>\n');
10    }
11 }
12
13 req = [
14     user:'Drew Wills',
15     getRemoteUser:{ return req['user'] }
16 ];
17
18 res = [
19     output:{ print it; } as ServletOutputStream,
20     getOutputStream:{ return res['output']; }
21 ];
22
23 new MySimpleServlet().doGet(req as HttpServletRequest, res as HttpServletResponse);
24 // prints...
25 // <body>
26 //   <h1>Hello Drew Wills!</h1>
27 // </body>
28
29
```

# Break



# Groovy Truth

- Groovy can coerce non-boolean objects into boolean values
  - Collections are **true** when they contain any values, otherwise **false**
  - Iterators and Enumerators are **true** when they contain further values, otherwise **false**
  - Regex Matchers are **true** when they contain at least one match, otherwise **false**
  - Strings, GStrings, and CharacterSequences are **true** when they contain any characters, otherwise **false**
  - Non-zero numbers are **true**; zero is **false**
  - Non-null object references are **true**; **null** is **false**

# Groovy Truth (cont.)

```
1 boolean coerceIt(Object o) {
2     return (Boolean) o;
3 }
4
5 // Collections
6 println coerceIt([]); // prints "false"
7 println coerceIt([msg:'Hello']); // prints "true"
8
9 // Iterators and Enumerators
10 println coerceIt([].entrySet().iterator()); // prints "false"
11 println coerceIt([msg:'Hello'].entrySet().iterator()); // prints "true"
12
13 // Regex Matchers
14 println coerceIt('foo' =~ /bar/); // prints "false"
15 println coerceIt('foobar' =~ /bar/); // prints "true"
16
17 // Strings, GStrings, and CharacterSequences
18 println coerceIt(''); // prints "false"
19 println coerceIt('Hello'); // prints "true"
20
21 // Numbers
22 println coerceIt(0); // prints "false"
23 println coerceIt(1); // prints "true"
24
25 // Object references
26 println coerceIt(null); // prints "false"
27 println coerceIt(new Date()); // prints "true"
28
29
```

# JDK Library Enhancements

- Groovy includes new features for dozens of commonly used classes in the JDK
  - <http://groovy.codehaus.org/groovy-jdk/>
- Examples
  - **String.center(Number)**
  - **List.first()** and **List.last()**
  - **Map.subMap(Collection)**



# JDK Library Enhancements (cont.)

- Perhaps a majority of added methods take a **Closure** as a parameter
  - For iteration...

```
1 def sum = 0;
2 [4, 5, 6].each {
3     sum += it;
4 };
5 println "The total is ${sum}";
6 // prints "The total is 15"
7
8 new File('some-file.txt').eachLine {
9     println it;
10 }; // prints...
11 // I hope everyone is
12 // learning a lot about
13 // Groovy and having fun!
14
15
```

# JDK Library Enhancements (cont.)

- For transformation...

```
1 println 'foobar'.replaceAll(/[a-f]/) {  
2     return it.toUpperCase();  
3 };  
4 // prints "FooBAR"  
5  
6 def map = [1:2, 3:4];  
7 println map.collect { k, v ->  
8     return "${k} + ${v} = ${k+v}";  
9 };  
10 // prints "[1 + 2 = 3, 3 + 4 = 7]"  
11  
12
```

# JDK Library Enhancements (cont.)

- Or for sorting...

```
1 def standingsGroupC = [  
2     'England':6,  
3     'United States':9,  
4     'Algeria':1,  
5     'Slovenia':1  
6 ];  
7  
8 println 'The standings for Group C are:'  
9 standingsGroupC.sort { o1, o2 ->  
10     return o2.getValue() - o1.getValue();  
11 }.each { k, v ->  
12     println " - ${k} at ${v} pts.";  
13 }; // prints...  
14 // The standings for Group C are:  
15 // - United States at 9 pts.  
16 // - England at 6 pts.  
17 // - Algeria at 1 pts.  
18 // - Slovenia at 1 pts.  
19  
20
```

# Metaprogramming

- a.k.a. *Monkey Patching*

*[...] a way to extend or modify the runtime code of dynamic languages [...] without altering the original source code*

Wikipedia

- Use caution

**Homer:** See Marge, they *could* deep-fry my shirt.

**Marge:** I didn't say they couldn't. I said you shouldn't.

# Metaprogramming (cont.)

- Meta Object Protocol
  - Every object in Groovy has a **MetaClass**
  - Every method call is dispatched through it
  - If you can alter an object's **MetaClass**, you can change its behavior

# Metaprogramming (cont.)

- Categories example

```
1 class Category {
2     static String foo(Object self) { return 'foo' };
3     static Object add(Object self, Object other) {
4         return self + other;
5     }
6 }
7
8 use (Category) {
9     println 1.foo(); // prints "foo"
10    println 1.add(2); // prints "3"
11    println 'one'.add(' monkey'); // prints "one monkey"
12 }
13
14
```

- Only works withing the `use {...}` block
- Not thread safe

# Metaprogramming (cont.)

- ExpandoMetaClass example

```
1 String.metaClass.scramble = {
2     StringBuilder rslt = new StringBuilder();
3     List chars = delegate.getChars();
4     Collections.shuffle(chars);
5     chars.each { rslt.append(it) };
6     return rslt.toString();
7 };
8
9 println 'Hello World!'.scramble();
10 // prints something random like...
11 // dolll rWeHo!
12 // llleoHd !rWo
13
14 String.metaClass.equals = {
15     return false;
16 };
17
18 println 'foo'.equals('foo'); // prints "false"!!!
19
20
21
```

# Puzzler Exercise



See [groovy-puzzler-exercise/README.txt](#)  
on usb drive



# Recurse All Files Exercise



Write a groovy script which recursively descends through a given directory and writes the files out. Suggest using `groovyConsole` or `groovsh`

# Groovy Tools & Frameworks

Working with Groovy

# Spring Installation

Add the following dependencies to your pom.xml

```
<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy-all</artifactId>
  <version>1.6.5</version>
  <type>jar</type>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.codehaus.groovy.maven.runtime</groupId>
  <artifactId>gmaven-runtime-1.6</artifactId>
  <version>1.0</version>
  <scope>compile</scope>
</dependency>
```

# Spring Installation for unit testing

Add the following to the `<build>` section of your `pom.xml`:

```
<plugin>
  <groupId>org.codehaus.groovy.maven</groupId>
  <artifactId>gmaven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
        <goal>testCompile</goal>
      </goals>
      <configuration>
        <targetBytecode>1.5</targetBytecode>
        <debug>true</debug>
      </configuration>
    </execution>
  </executions>
</plugin>
```

# Groovy-Eclipse Plugin

- Version 2.0 released January 15th, 2010
- Update Site

<http://dist.springsource.org/release/GRECLIPSE/e3.5/>

- Documentation

<http://docs.codehaus.org/x/uAo>

- Works with Eclipse 3.5 Galileo; a release for 3.4 is also available (but not maintained)



# Grails

- Web development framework for Groovy & Java based on Ruby on Rails
- Features
  - Object Relational Mapping (ORM) based on Hibernate
  - Groovy Server Pages (GSM)
  - Spring MVC Controller Layer
  - Command line script environment built on Gant
  - Embedded Tomcat
  - Spring dependency injection
  - Spring **MessageSource** for i18n
  - Spring transactions



# Groovy + Testing

A great match

# Groovy Unit Testing

Groovy simplifies JUnit testing

- Groovy ignores access specifiers
  - No need to 'expose' methods just to unit test!
- JUnit is built into groovy runtime
  - Trivial to build/run
- Groovy unit tests scriptable with Ant/Maven
- Groovy provides Groovy Mocks



# “Groovy Mock”

- Using closures instead of mocks
- Using Maps and Expandos instead of mocks
- Mocking of static methods
- MockFor and StubFor
  - Used for testing classes in isolation

# But what about test framework x?

- EasyMock
- Instinct
- JBehave
- JDummy
- JMockit
- Popper
- RMock
- TestNG

EASYMOCK



jMock

JBehave

# Fun with files Exercise



See [voting-machine-exercise/README.txt](#)

on usb drive

# Questions?



Drew Wills

[drew@unicon.net](mailto:drew@unicon.net)

Lennard Fuller

[lfuller@unicon.net](mailto:lfuller@unicon.net)