Groovy For Java Programmers

Venkat Subramaniam

venkats@agiledeveloper.com

http://www.agiledeveloper.com/download.aspx

Abstract

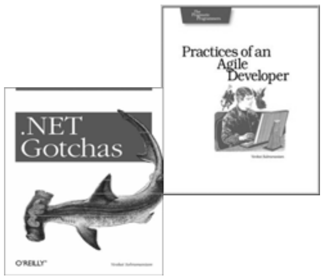
Abstract Object-oriented scripting languages, or agile dynamic languages, as some like to call those, are gaining programmers' attention. Groovy bring this excitement to the Java platform with its ability to generate byte code. You can use Groovy instead of Java for some parts of your application. By learning it, you can switch between the languages where you consider fit.

In this session we will learn what Groovy is. We will take an example driven approach to look at interesting features. We will see how a piece of code you would write in Java can be written, elegantly, using Groovy. In addition to the current features, we will also discuss the state of the language and tools.

About the Speaker *Dr. Venkat Subramaniam*, founder of Agile Developer, Inc., has trained and mentored thousands of software developers in the US, Canada, and Europe. He has significant experience in architecture, design, and development of software applications. Venkat helps his clients effectively apply and succeed with agile practices on their software projects, and speaks frequently at conferences.

He is also an adjunct faculty at the University of Houston (where he received the 2004 CS department teaching excellence award) and teaches the professional software developer series at Rice University School of continuing studies.

Venkat has been a frequent speaker at No Fluff Just Stuff Software Symposium since Summer 2002.



Groovy for Java Programmers

- **What's Groovy**
- Tools
- Java and Groovy
- Collections
- Operator Overloading
- Groovy String
- Closure
- Regex
- Groovy Markup
- GPath
- Groovy SQL
- Conclusion

What's Groovy

- Dictionary: Marvelous, Excellent; HIP
- Power of scripting language into Java platform
  - Dynamic, agile, OO
  - Derives strengths from Smalltalk, Python, Ruby
- Alternative to Java for small/medium size projects
- Serves to write small additional tasks on a project
- Great to write unit tests with
- Can generate Java byte code which can be used with Java code

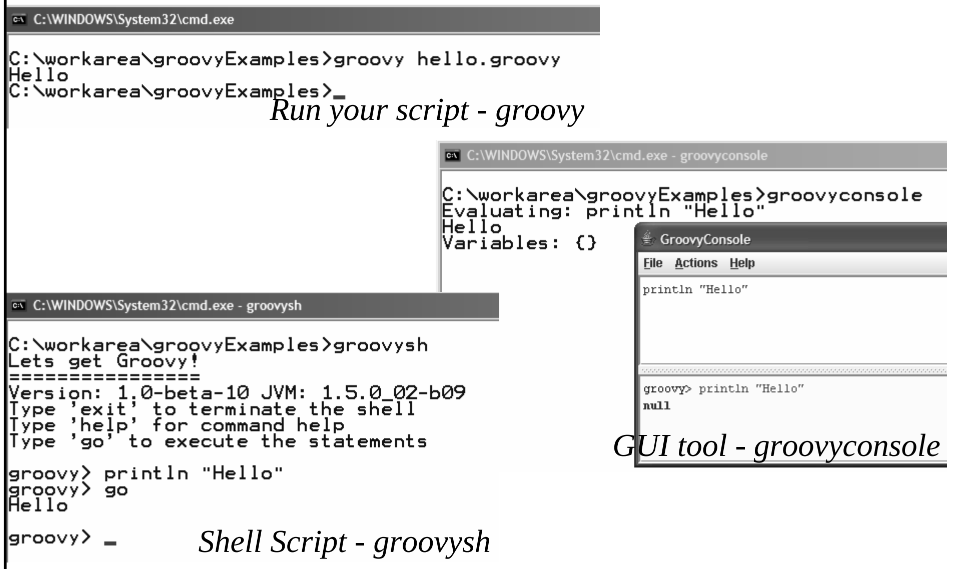
History

- August 2003
  - James Strachan and Bob McWhirter
- JSR-241
- April 2005 JSR-01
- June 2005 JSR-02
- August 2005 JSR-03
- Current 1.0 beta 10

Groovy for Java Programmers

- What’s Groovy
- **Tools**
- Java and Groovy
- Collections
- Operator Overloading
- Groovy String
- Closure
- Regex
- Groovy Markup
- GPath
- Groovy SQL
- Conclusion

Tools



```
C:\WINDOWS\System32\cmd.exe

C:\workarea\groovyExamples>groovy hello.groovy
Hello
C:\workarea\groovyExamples>_
```

Run your script - groovy

```
C:\WINDOWS\System32\cmd.exe - groovyconsole

C:\workarea\groovyExamples>groovyconsole
Evaluating: println "Hello"
Hello
Variables: {}
```

```
GroovyConsole
File  Actions  Help
println "Hello"

groovy> println "Hello"
null
```

GUI tool - groovyconsole

```
C:\WINDOWS\System32\cmd.exe - groovysh

C:\workarea\groovyExamples>groovysh
Lets get Groovy!
================
Version: 1.0-beta-10 JVM: 1.5.0_02-b09
Type 'exit' to terminate the shell
Type 'help' for command help
Type 'go' to execute the statements

groovy> println "Hello"
groovy> go
Hello

groovy> _
```

Shell Script - groovysh

Groovy for Java Programmers

- What's Groovy
- Tools
- **Java and Groovy**
- Collections
- Operator Overloading
- Groovy String
- Closure
- Regex
- Groovy Markup
- GPath
- Groovy SQL
- Conclusion

Hello World in Java & Groovy

- Code less, do more
- Java (and Groovy):

```
// This Java program is a groovy program as well.
public class Hello
{
        public static void main(String[] args)
        {
                System.out.println("Hello");
        }
}
```

GOJ

- Groovy

```
println 'Hello'
```

- Not all Java code is Groovy code
  – Does not handle inner classes, Generic code

Variables

- Dynamic typing (though you may type variables if you like)

```
x = 1
println x

x = new Date()
println x

x = "test"
println x
```

```
1
Mon Sep 26 16:57:03 CDT 2005
test
```

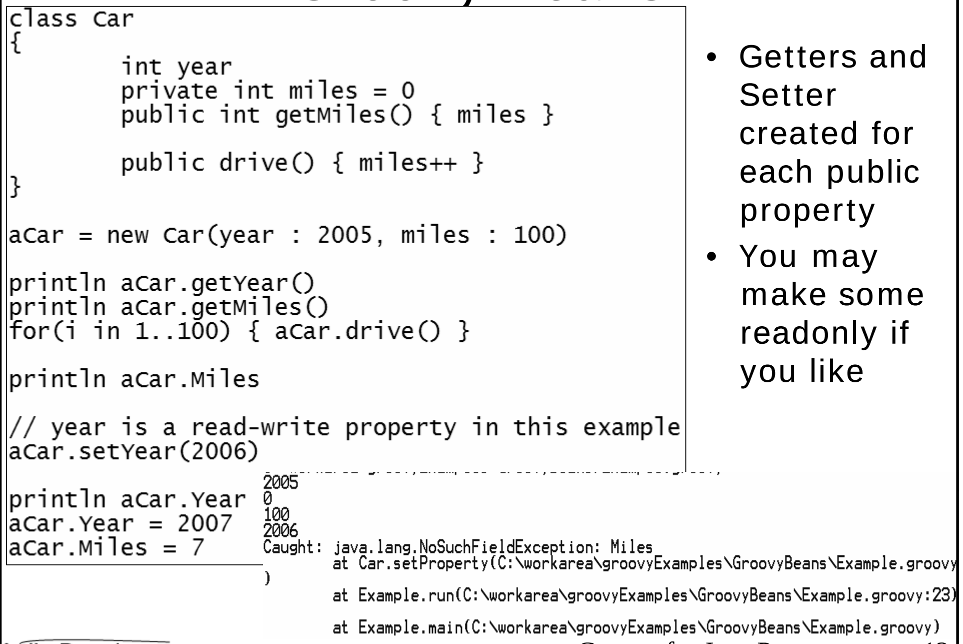
Compiling Groovy

```
Compiled from "Hello.groovy"
public class Hello extends groovy.lang.Script{
public static java.lang.Long __timeStamp;
static java.lang.Class class$0;
static java.lang.Class class$org$codehaus$groovy$runtime$InvokerHelper;
public Hello();
  Code:
   0:   aload_0
   1:   invokespecial   #12; //Method groovy/lang/Script."<init>":()V
   4:   return
public Hello(groovy.lang.Binding);
  Code:
   0:   aload_0
   1:   invokespecial   #12; //Method groovy/lang/Script."<init>":()V
   …
   9:   return
public static void main(java.lang.String[]);
  Code:
   0:   iconst_2
   1:   anewarray       #20; //class java/lang/Object
                …
```

•Groovy can be scripted
•If you like, compile to Java byte code

•groovyc Hello.groovy, where Hello.groovy contains println 'Hello'

Groovy Beans



```
class Car
{
        int year
        private int miles = 0
        public int getMiles() { miles }

        public drive() { miles++ }
}

aCar = new Car(year : 2005, miles : 100)

println aCar.getYear()
println aCar.getMiles()
for(i in 1..100) { aCar.drive() }

println aCar.Miles

// year is a read-write property in this example
aCar.setYear(2006)

println aCar.Year
aCar.Year = 2007
aCar.Miles = 7
```

```
2005
0
100
2006
Caught: java.lang.NoSuchFieldException: Miles
        at Car.setProperty(C:\workarea\groovyExamples\GroovyBeans\Example.groovy
)
        at Example.run(C:\workarea\groovyExamples\GroovyBeans\Example.groovy:23)
        at Example.main(C:\workarea\groovyExamples\GroovyBeans\Example.groovy)
```

- Getters and Setter created for each public property
- You may make some readonly if you like

Groovy for Java Programmers

- What's Groovy
- Tools
- Java and Groovy
- **Collections**
- Operator Overloading
- Groovy String
- Closure
- Regex
- Groovy Markup
- GPath
- Groovy SQL
- Conclusion

Using List

GOJ

```
import java.util.*;

public class Example
{
        public static void main(String[] args)
        {
                List<Integer> lst = new ArrayList<Integer>();
                lst.add(1);
                lst.add(2);

                int total = 0;

                for(int val : lst)
                {
                        total += val;
                }

                System.out.println(total);
        }
}
```

```
lst = [1, 2]
total = 0

for(val in lst)
{
        total += val
}

println total
```

Accessing List Elements

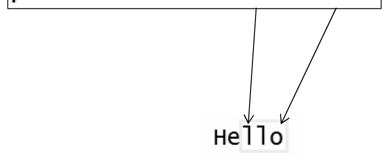
```
lst = [1, 'hello', 'a', 3.2]

println "Size of lst is ${lst.size()}"
println "Element 0 is ${lst.get(0)}"
println "Element 2 is ${lst[2]}"

lst[3] = 'test'
println lst

lst[2] = ['how', 'about', 'this', 1]
println lst

lst.putAt(1, 'replaced')
println lst

lst.putAt(1..0, 'inserted')
println lst

lst.putAt(2..1, ['some', 'more'])
println lst

print 'Result of flatten:'
println lst.flatten()
```

```
Size of lst is 4
Element 0 is 1
Element 2 is a
[1, hello, a, test]
[1, hello, [how, about, this, 1], test]
[1, replaced, [how, about, this, 1], test]
[1, inserted, replaced, [how, about, this, 1], test]
[1, inserted, some, more, replaced, [how, about, this, 1], test]
Result of flatten:[1, inserted, some, more, replaced, how, about, this, 1, test]
```

Range

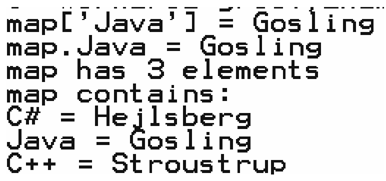
```
str = "Hello"

println str[0]
println str[1..4]
println str[2..0]
println str[2..-1]
```

```
H
ello
leH
llo
```



Map

```
map = ['C++' : 'Stroustrup', 'Java' : 'Gosling', 'C#' : 'Hejlsberg']

println "map['Java'] = ${map['Java']}"

println "map.Java = ${map.Java}"
println "map has ${map.size()} elements"

println "map contains:"
map.each {key, value | println "${key} = ${value}"}
```



```
map['Java'] = Gosling
map.Java = Gosling
map has 3 elements
map contains:
C# = Hejlsberg
Java = Gosling
C++ = Stroustrup
```

Groovy for Java Programmers

- What’s Groovy
- Tools
- Java and Groovy
- Collections
- **Operator Overloading**
- Groovy String
- Closure
- Regex
- Groovy Markup
- GPath
- Groovy SQL
- Conclusion

Operator Overloading

- Groovy provides overloaded operators and mapping to several Java methods

From http://groovy.codehaus.org/Operator+Overloading

- Operator | Method
  a + b | a.plus(b)
  a - b | a.minus(b)
  a * b | a.multiply(b)
  a / b | a.divide(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.equals(b)
  a != b | ! a.equals(b)
  a === b | a == b in Java
  (i.e. a and b refer to same object instance)
  a <=> b | a.compareTo(b)
  a > b | a.compareTo(b) > 0
  a >= b | a.compareTo(b) >= 0
  a < b | a.compareTo(b) < 0
  a <= b | a.compareTo(b) <= 0

Operator Overloading

```
myVector = new VenkatsVector()
println myVector

anotherVector = myVector + myVector
println anotherVector

class VenkatsVector
{
        vals = [1, 2, 3]

        VenkatsVector plus(VenkatsVector v)
        {
                for(i in 1..vals.size())
                {
                        vals[i-1] += v.vals[i-1]
                }

                this
        }

        public String toString()
        {
                vals
        }
}
```

```
[1, 2, 3]
[2, 4, 6]
```

```
c = 'A'
print "Char next to ${c} is:"
c++
println c

str = 'working hard'
println str + '?'
str += 'ly!'
str -= 'working '
println str
```

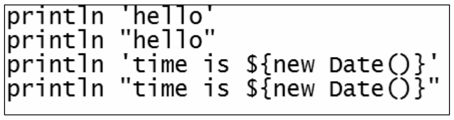
```
Char next to A is:B
working hard?
hardly!
```

Groovy for Java Programmers

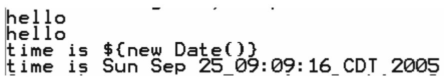
- What’s Groovy
- Tools
- Java and Groovy
- Collections
- Operator Overloading
- **Groovy String**
- Closure
- Regex
- Groovy Markup
- GPath
- Groovy SQL
- Conclusion

"" vs '' (Double vs. Single Quotes)

- '' is a simple string
- Expressions within "" are evaluated



```
println 'hello'
println "hello"
println 'time is ${new Date()}'
println "time is ${new Date()}"
```

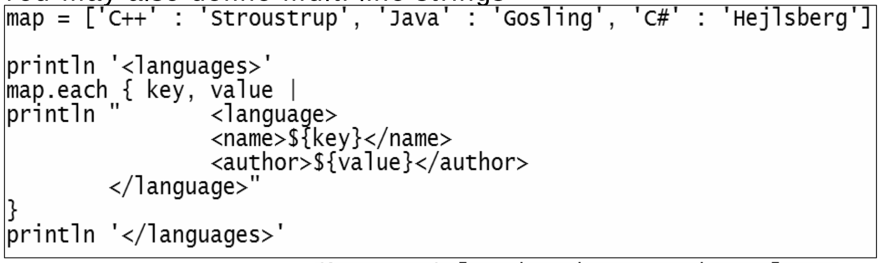


```
hello
hello
time is ${new Date()}
time is Sun Sep 25 09:09:16 CDT 2005
```

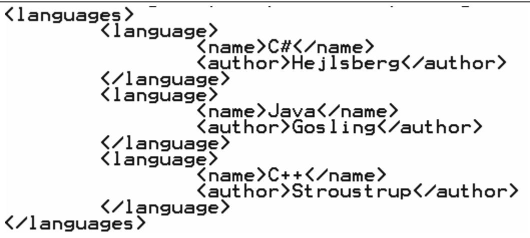
- Better to use ' if you don’t have expressions

Groovy Strings - GStrings

- Strings in Groovy are pretty powerful
  - Java String with things added
  - As you saw, expressions are evaluated in creating the string
- You may also define multi line strings



```
map = ['C++' : 'Stroustrup', 'Java' : 'Gosling', 'C#' : 'Hejlsberg']

println '<languages>'
map.each { key, value |
println "        <language>
                <name>${key}</name>
                <author>${value}</author>
        </language>"
}
println '</languages>'
```



```
<languages>
        <language>
                <name>C#</name>
                <author>Hejlsberg</author>
        </language>
        <language>
                <name>Java</name>
                <author>Gosling</author>
        </language>
        <language>
                <name>C++</name>
                <author>Stroustrup</author>
        </language>
</languages>
```

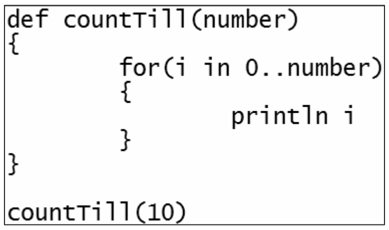
Wait till you see MarkupBuilder later

Groovy for Java Programmers

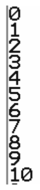
- What’s Groovy
- Tools
- Java and Groovy
- Collections
- Operator Overloading
- Groovy String
- **Closure**
- Regex
- Groovy Markup
- GPath
- Groovy SQL
- Conclusion

Writing a function

- def a function just like you would in Java
- No need to return, the value of last statement is returned



```
def countTill(number)
{
        for(i in 0..number)
        {
                println i
        }
}

countTill(10)
```

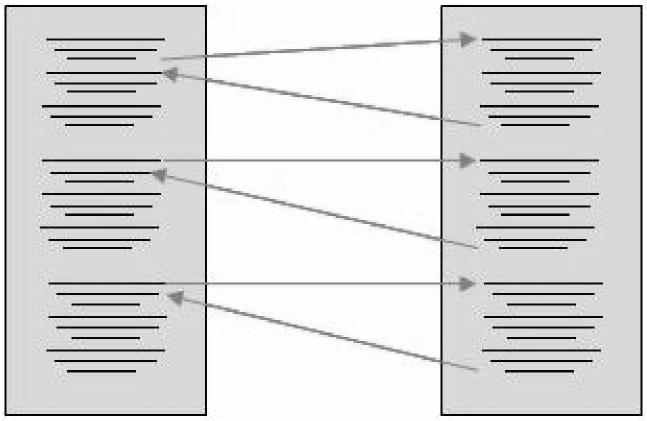


```
0
1
2
3
4
5
6
7
8
9
10
```

Closures

- An anonymous chunk of code you can invoke
  - take arguments
  - return a value
  - reference and use variables declared in its surrounding scope
- Some what like anonymous inner classes in Java, but more powerful and convenient
- How is this different from a method?
- You directly call a method
- Closure is really cool, it is used for the method you call to call back
- Now we are talking about coroutines

Coroutines

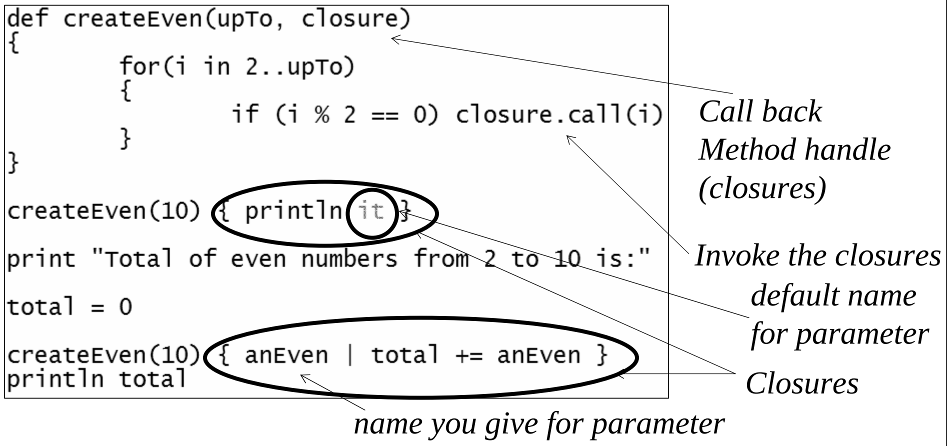


Coroutines with closures

- I have a method that creates even numbers until a given limit
- What do you want to do with these even numbers?
- That is up to you, it depends on what you want to do at the place of call

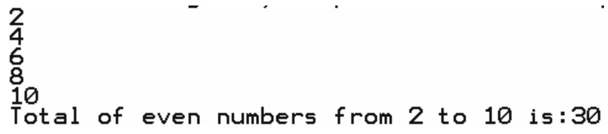
- So, my method is going to create even numbers, but will yield to your code so you can do something with it

Closures



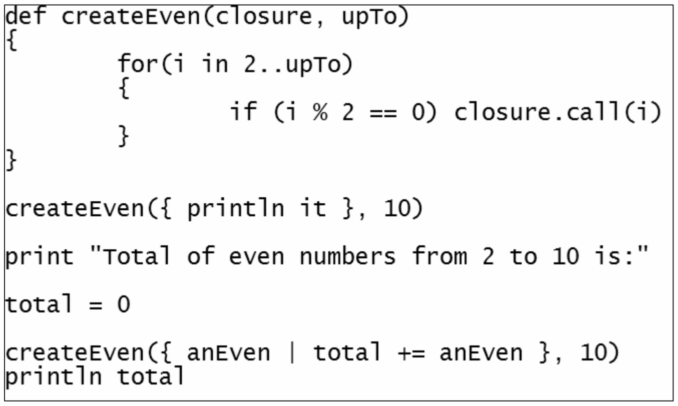
```
def createEven(upTo, closure)
{
        for(i in 2..upTo)
        {
                if (i % 2 == 0) closure.call(i)
        }
}

createEven(10) { println it }

print "Total of even numbers from 2 to 10 is:"

total = 0

createEven(10) { anEven | total += anEven }
println total
```

(lots of syntax sugar)



```
2
4
6
8
10
Total of even numbers from 2 to 10 is:30
```

Closures As Parameter



```
def createEven(closure, upTo)
{
        for(i in 2..upTo)
        {
                if (i % 2 == 0) closure.call(i)
        }
}

createEven({ println it }, 10)

print "Total of even numbers from 2 to 10 is:"

total = 0

createEven({ anEven | total += anEven }, 10)
println total
```

When closure is last parameter,
it can be outside () as shown in previous page

Closures and Collections

```
lst = [1, 2, 3, 8]

print "Elements in lst are:"

lst.each { print "${it} " }

println ""
print "List with each element incremented:"
println lst.collect { it + 1 }

found = lst.find { println "find sent ${it}"; it == 3 }
println "found ${found}"

println "lst with elements > 2: ${lst.findAll { it > 2 }}"
println "Total of elements: " +
        lst.inject(0) { carryOver, item | carryOver + item }
println "Elements joined with ~ : " + lst.join('~')
```

Calls closures for ***each*** *element*

Calls for each element and collects result

Returns first element for which closures returns true

injects parameter before first element, passes result from closures to next invocation

Returns all elements for which closures returns true

Concatenates elements with the string

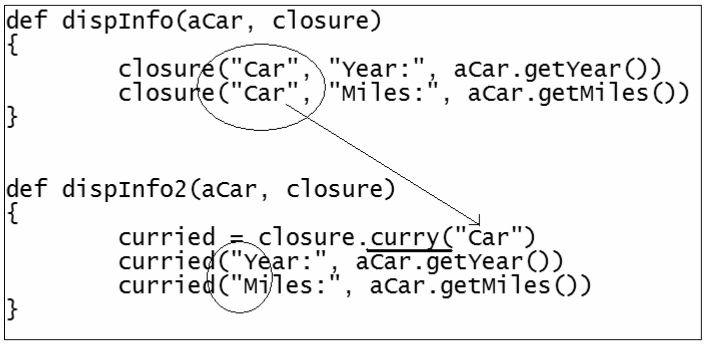
Accessing File

```
new java.io.File('quote.txt').eachLine {
        println it
}
```

```
"I'm gonna make him an offer he can't refuse."
Don Corleone said to Johnny Fontane about Woltz.
```

Curried Closure

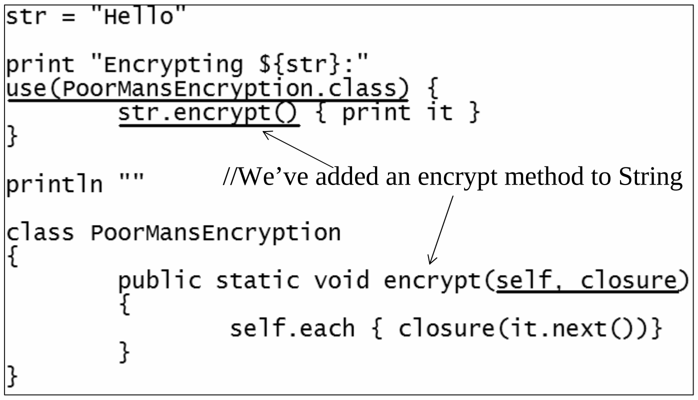
- For convenience, you may fix value for one or more arguments to a closure
- This is called currying
- The arguments you fixed will be automatically sent when you call the curried closure



```
def dispInfo(aCar, closure)
{
        closure("Car", "Year:", aCar.getYear())
        closure("Car", "Miles:", aCar.getMiles())
}

def dispInfo2(aCar, closure)
{
        curried = closure.curry("Car")
        curried("Year:", aCar.getYear())
        curried("Miles:", aCar.getMiles())
}
```

Adding Methods

- You can add method to class using a use directive
- The method, static, must take at least two parameters
  - first is type to which you are adding method
  - last is the closure to call



```
str = "Hello"

print "Encrypting ${str}:"
use(PoorMansEncryption.class) {
        str.encrypt() { print it }
}

println ""        //We've added an encrypt method to String

class PoorMansEncryption
{
        public static void encrypt(self, closure)
        {
                self.each { closure(it.next())}
        }
}
```

```
Encrypting Hello:Ifmmp
```

Groovy for Java Programmers

- What’s Groovy
- Tools
- Java and Groovy
- Collections
- Operator Overloading
- Groovy String
- Closure
- **Regex**
- Groovy Markup
- GPath
- Groovy SQL
- Conclusion

Regex

- Exceptional pattern matching ability
  - uses java.util.regex library
- =~ for comparison
  - Pattern matching instead of equality

```
def check(str, closure)
{
        if (str =~ '(G|g)roovy')
                closure.call(str, true);
        else
                closure.call(str, false);
}

dispClosure = { str, res |
        println "Match found for ${str}?: ${res}" }

check('Groovy', dispClosure)
check('groovy', dispClosure)
check('Gooovy', dispClosure)

println "Result of 'Groovy'.split('o+')"
'Groovy'.split('o+').each { println it }
```

```
Match found for Groovy?: true
Match found for groovy?: true
Match found for Gooovy?: false
Result of 'Groovy'.split('o+')
Gr
vy
```

Groovy for Java Programmers

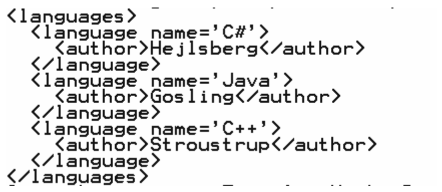
- What's Groovy
- Tools
- Java and Groovy
- Collections
- Operator Overloading
- Groovy String
- Closure
- Regex
- **Groovy Markup**
- GPath
- Groovy SQL
- Conclusion

Groovy Markup

- Provide markup for several things
  - XML – NodeBuilder
  - HTML – MarkupBuilder
  - SAX – SAXBuilder
  - W3C DOM – DOMBuilder
  - Ant tasks – AntBuilder
  - Swing – SwingBuilder

Markup Builder

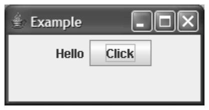
```
import groovy.xml.MarkupBuilder

map = ['C++' : 'Stroustrup', 'Java' : 'Gosling', 'C#' : 'Hejlsberg']

markupBldr = new MarkupBuilder()

xml = markupBldr.languages {
        for(entry in map)
        {
                language(name : entry.key) {
                        author (entry.value)
                }
        }
}
```

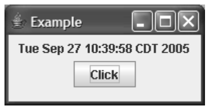


```
<languages>
  <language name='C#'>
    <author>Hejlsberg</author>
  </language>
  <language name='Java'>
    <author>Gosling</author>
  </language>
  <language name='C++'>
    <author>Stroustrup</author>
  </language>
</languages>
```

SwingBuilder

```
// Closure
setLabel = { lbl | lbl.setText(new Date().toString()) }

swingBldr = new groovy.swing.SwingBuilder()


frame = swingBldr.frame(
        title : "Example",
        size:[200, 100],
        layout: new java.awt.FlowLayout(),
        defaultCloseOperation:javax.swing.WindowConstants.EXIT_ON_CLOSE)

        myLabel = label(text: "Hello")
        button(text : 'Click', actionPerformed: {  setLabel(myLabel) })
}

frame.show()
```





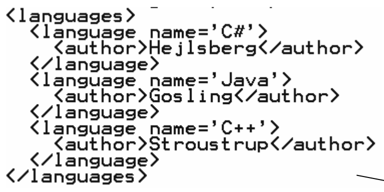
Groovy for Java Programmers

- What’s Groovy
- Tools
- Java and Groovy
- Collections
- Operator Overloading
- Groovy String
- Closure
- Regex
- Groovy Markup
- **GPath**
- Groovy SQL
- Conclusion

GPath

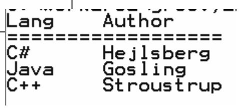
- Based on XPath for XML
- Expression language for any tree structured data
- a.b.c means any <c> element inside <b> which is inside <a>
- a['@something'] means attribute something of <a>
- One example if parsing XML

Parsing XML



```
<languages>
  <language name='C#'>
    <author>Hejlsberg</author>
  </language>
  <language name='Java'>
    <author>Gosling</author>
  </language>
  <language name='C++'>
    <author>Stroustrup</author>
  </language>
</languages>
```

```
languages = new XmlParser().parse("input.xml")

println "Lang\tAuthor"
println "=================="

languages.each {
        print it['@name']
        print '\t'
        println it.author[0].text() }
```



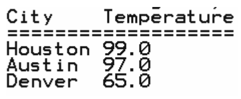
```
Lang    Author
==================
C#      Hejlsberg
Java    Gosling
C++     Stroustrup
```

Groovy for Java Programmers

- What’s Groovy
- Tools
- Java and Groovy
- Collections
- Operator Overloading
- Groovy String
- Closure
- Regex
- Groovy Markup
- GPath
- **Groovy SQL**
- Conclusion

Groovy SQL

```
// I've added Temperatures.xls as
//a ODBC datasource named 'Temperatures'

import groovy.sql.Sql

sql = Sql.newInstance(
        'jdbc:odbc:Temperatures', '', '',
        'sun.jdbc.odbc.JdbcOdbcDriver')

println 'City\tTemperature'
println '=================='

sql.eachRow('select * from [TemperatureValues$]') {
                println "${it.City}\t${it.temperature}" }
```



```
City    Temperature
==================
Houston 99.0
Austin  97.0
Denver  65.0
```

Quiz Time

Groovy for Java Programmers

- What's Groovy
- Tools
- Java and Groovy
- Collections
- Operator Overloading
- Groovy String
- Closure
- Regex
- Groovy Markup
- GPath
- Groovy SQL
- **Conclusion**

Conclusion

- +
  - Groovy brings the power of scripting languages–dynamic and OO–to Java platform
  - You can mix Java API with elegant constructs of Groovy
  - Ideal to try out for small projects
    - Little tasks you need to work on
    - Unit tests can be written using Groovy
- -
  - Not complete yet
  - Not all Java code is Groovy code yet
    - Inner classes, for(int i = 0; i < count; i++), code with generics in it, …
  - Debugging is very hard
  - Performance of Groovy code is slower than Java

References

1. http://groovy.codehaus.org
2. http://groovy.codehaus.org/Language+Guide
3. http://groovy.codehaus.org/groovy-jdk.html
4. Download examples/slides from

http://www.agiledeveloper.com/download.aspx

Please fill out your evaluations!