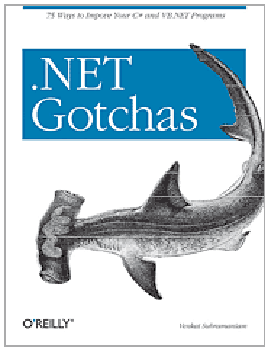
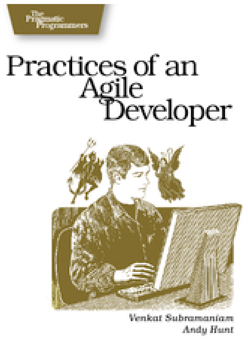
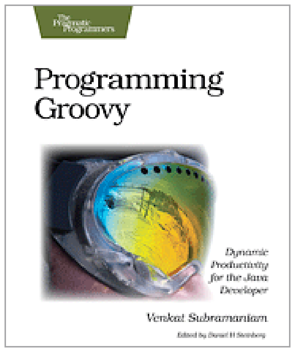
BLENDING JAVA WITH DYNAMIC LANGUAGES

```
speaker.identity {
  name        'Venkat Subramaniam'
  company     'Agile Developer, Inc.'
  credentials 'Programmer', 'Author', 'Trainer'
  blog        'http://agiledeveloper.com/blog'
  email       'venkats@agiledeveloper.com'
}
```







Abstract

- The last several years have brought us some exciting advances in the capability and strength of the Java platform. At the same time, developers are increasingly excited about the productivity gains promised by the use of dynamic languages. The good news is that it is possible to get the best of both worlds—to take advantage of dynamic languages and leverage your knowledge of and investments in the Java Platform at the same time!
- In this presentation we will take an in-depth look at mixing dynamic languages and Java in the same application. We'll first look at it from the perspective of full interaction, and explore some idiomatic differences in interaction. Then, from an application development perspective, we'll discuss how these can help in areas like Rules Engine, DSLs, and Meta Programming.

Java: The Language and the Platform

- Java started out as a powerful, yet simple language
- Through it we realized WORA—Write Once Run Anywhere
- 'C' like language with Automatic Garbage Collection
- Powerful set of API and libraries
- Strong community of passionate developers and innovators
- Now we realize, the real strength of Java is not in the language
- It's in the platform

Dynamic Languages

- Dynamic Languages have been around for a long time
- Facilitate ease of metaprogramming, building DSLs, ... leading to higher productivity
- There is renewed interest in this area
- Why?
  - Machines have gotten faster
  - More Availability—community based development
  - Awareness of test driven development
  - Excitement from killer apps

Java Language*s*

- Java was once this single language on multiple platforms
- .NET was multiple languages on a single platform
- Now Java has become a true multiple languages on multiple platforms

Groovy JavaScript

JRuby Jython

Java Bytecode

Java Jaskell

Multi–Language Means What?

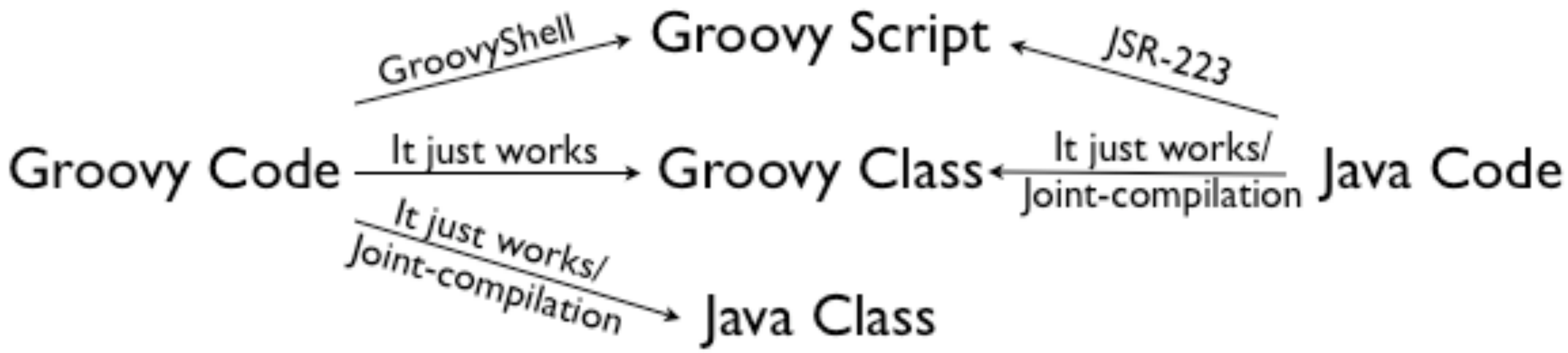
- Compiling from higher level languages to bytecode is not new
- But, *multi–language* means full interoperability with constructs created in different languages
- Can you inherit from a class created in another language?
- Can you associate or aggregate classes created in another language?
- Can you intermix them without major restrictions?

Why Mix Dynamic Languages?

- Dynamic Languages bring power of Metaprogramming and DSL to the table
- Can improve your productivity
- You can allow your users to be more expressive
- You can use it for Rule Specification in Rule Engines
- You can let your program evolve
- You can take dynamic decisions based on certain input or application state

API for Interoperability

- Languages to Java API/JDK
  - Language specific facilities—different languages handle this differently
  - For example, here are options in Groovy

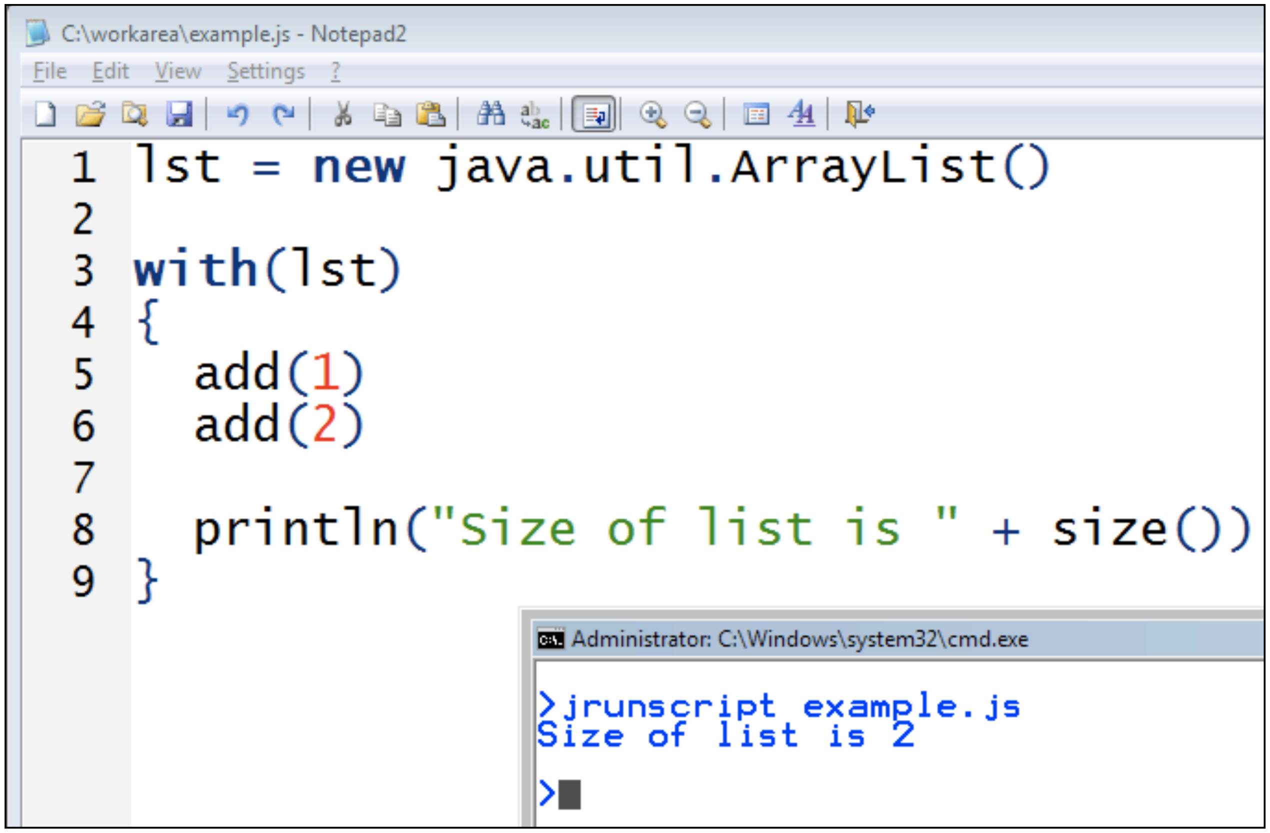


Source: "Programming Groovy: Dynamic Productivity for the Java Developer"

- Java to other languages
  - JSR–223 is a standard API for language interoperability
  - Useful to call from Java into other languages

JavaScript to Java

- You can use Java API/JDK from within JavaScript


C:\workarea\example.js - Notepad2
File Edit View Settings ?

```
lst = new java.util.ArrayList()

with(lst)
{
   add(1)
   add(2)

   println("Size of list is " + size())
}
```

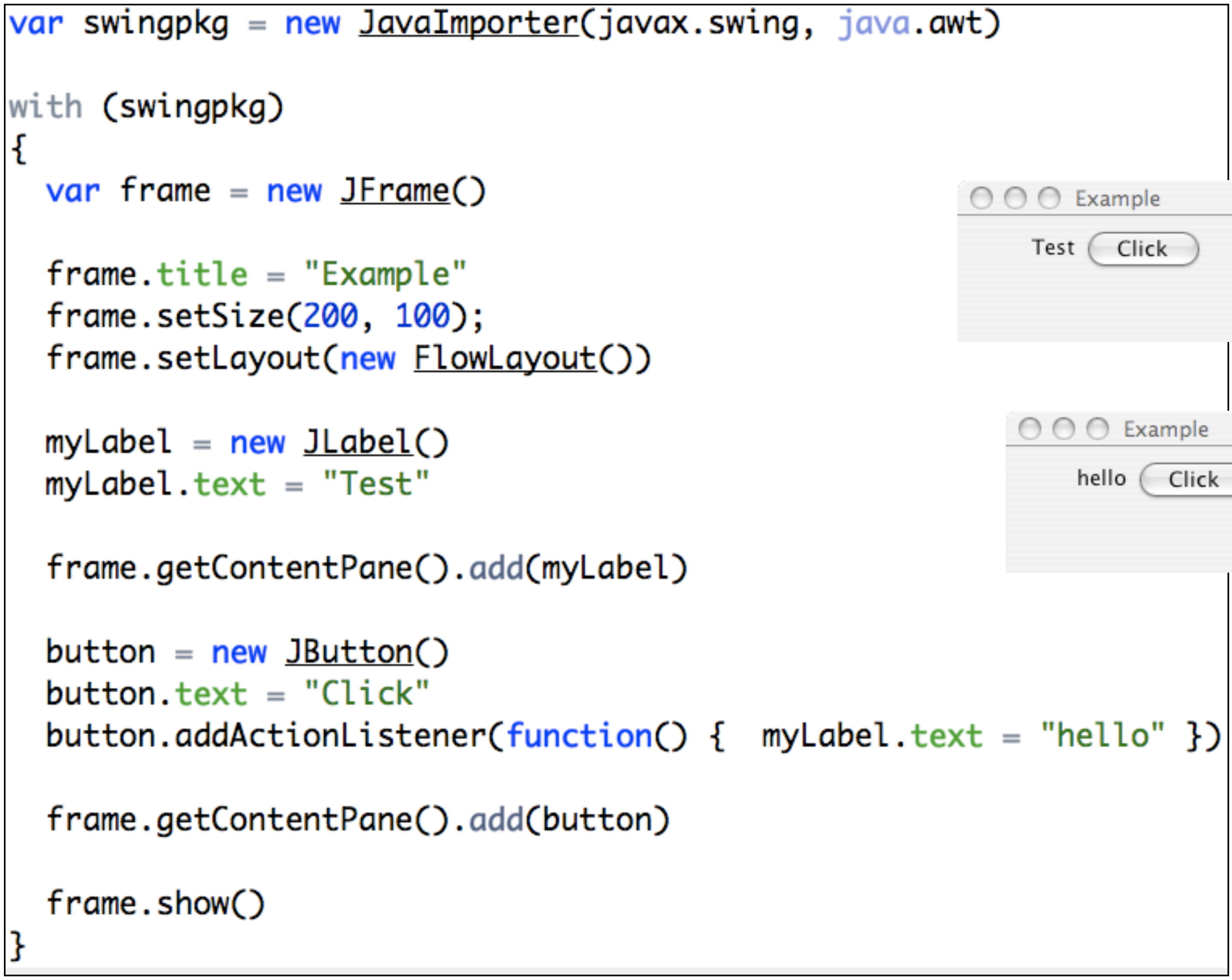
Administrator: C:\Windows\system32\cmd.exe

```
>jrunscript example.js
Size of list is 2

>
```

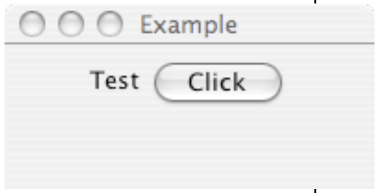
Idiomatic Difference

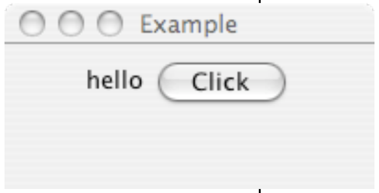
- One of the real fun in mixing languages is enjoying the idiomatic differences
- It is about calling Java API but using syntactic sugar and facilities of dynamic languages
- Reduces code size, gives you productivity

Building Swing App using JavaScript



```
var swingpkg = new JavaImporter(javax.swing, java.awt)

with (swingpkg)
{
  var frame = new JFrame()

  frame.title = "Example"
  frame.setSize(200, 100);
  frame.setLayout(new FlowLayout())

  myLabel = new JLabel()
  myLabel.text = "Test"

  frame.getContentPane().add(myLabel)

  button = new JButton()
  button.text = "Click"
  button.addActionListener(function() {  myLabel.text = "hello" })

  frame.getContentPane().add(button)

  frame.show()
}
```





Building Swing App using Groovy

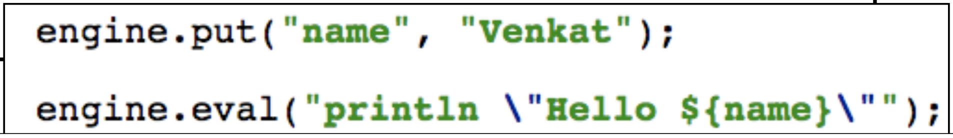
```
bldr = new groovy.swing.SwingBuilder()

frame = bldr.frame (title: 'Test', size:[200, 300], layout: new java.awt.FlowLayout())
{
  label = label(text: 'Test')
  button = button(text: 'Click me',
    actionPerformed: { label.text = new Date().toString() })
}

frame.show()
```

Calling JavaScript from Java

- Java allows you to call into scripts using JSR-223
- ScriptEngineManager allows you to query for and fetch ScriptEngines
- Once you obtain a ScriptEngine, use eval to execute any script

Calling JavaScript from Java...

```
package com.agiledeveloper;

import javax.script.*;

public class Script
{
	public static void main(String[] args)
	{
		try
		{
			ScriptEngineManager scriptManager = new ScriptEngineManager();

			ScriptEngine engine = scriptManager.getEngineByName("groovy");

			engine.eval("println 'Hello from Groovy'");
		}
		catch(ScriptException ex)
		{
			System.out.println("Error in scripting: " + ex);
		}
```



```
engine.put("name", "Venkat");

engine.eval("println \"Hello ${name}\"");
```

Invocable Interface

- Allows you to invoke functions and methods

```
package com.agiledeveloper;
import javax.script.*;

public class Script   {
        public static void main(String[] args) {
                try {
                        ScriptEngineManager scriptManager = new ScriptEngineManager();

                        ScriptEngine engine = scriptManager.getEngineByName("groovy");

                        engine.eval(
                "def count(val) { for (i in 1..val) { println i }; return 'Thank you for calling' }");

                        Invocable invocable = (Invocable) engine;

                        Object result = invocable.invokeFunction("count", 5);

                        System.out.println("Result from invocation is " + result);
                }
                catch(Exception ex) ...
```

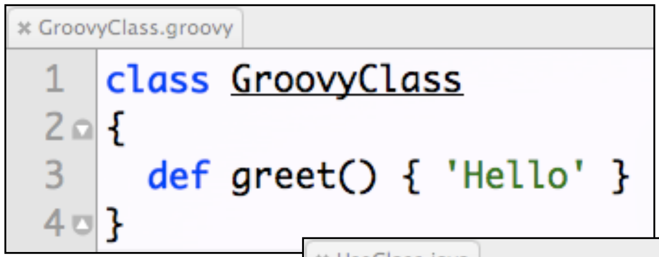
Compilable Interface

- If you’re going to make repeated calls to an interface you can ask it to be pre-compiled
- Provides more efficiency

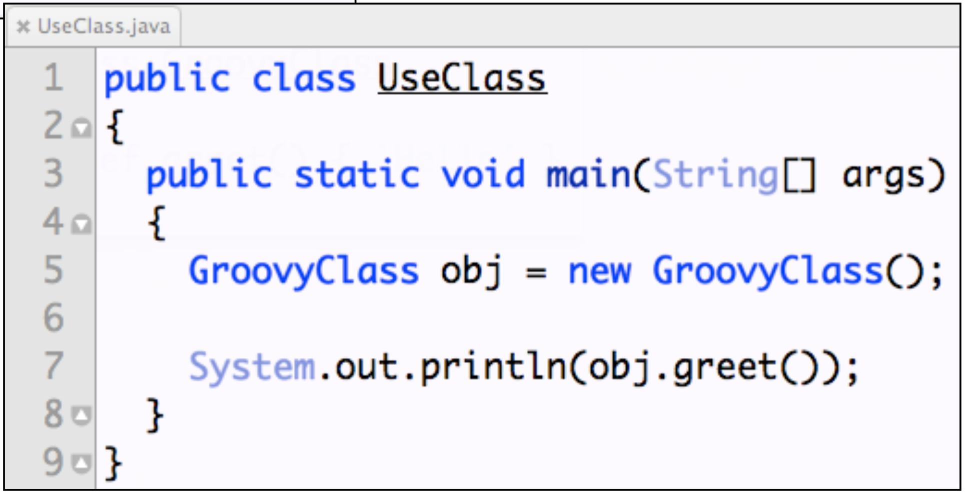
Groovy Simplifies this

- To call into Groovy from Java you don’t have to use JSR–223 unless you want to use script as is (without compilation)
- Groovy provides *joint* compilation
- You can compile Groovy code into Java bytecode and use it like any other Java code

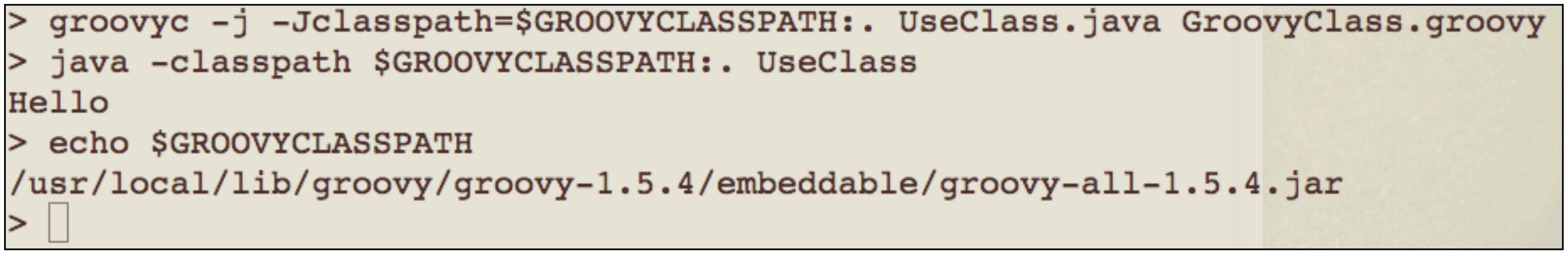
Using Groovy with Java


GroovyClass.groovy

```groovy
class GroovyClass
{
  def greet() { 'Hello' }
}
```

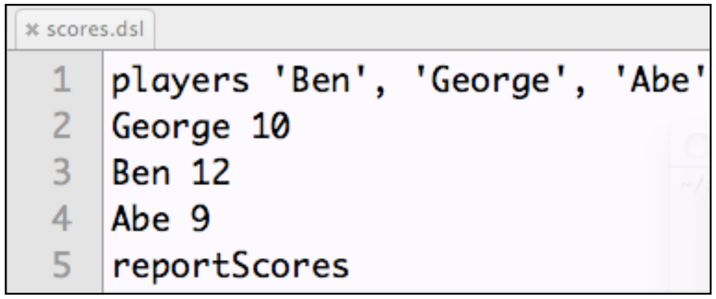

UseClass.java

```java
public class UseClass
{
  public static void main(String[] args)
  {
    GroovyClass obj = new GroovyClass();

    System.out.println(obj.greet());
  }
}
```

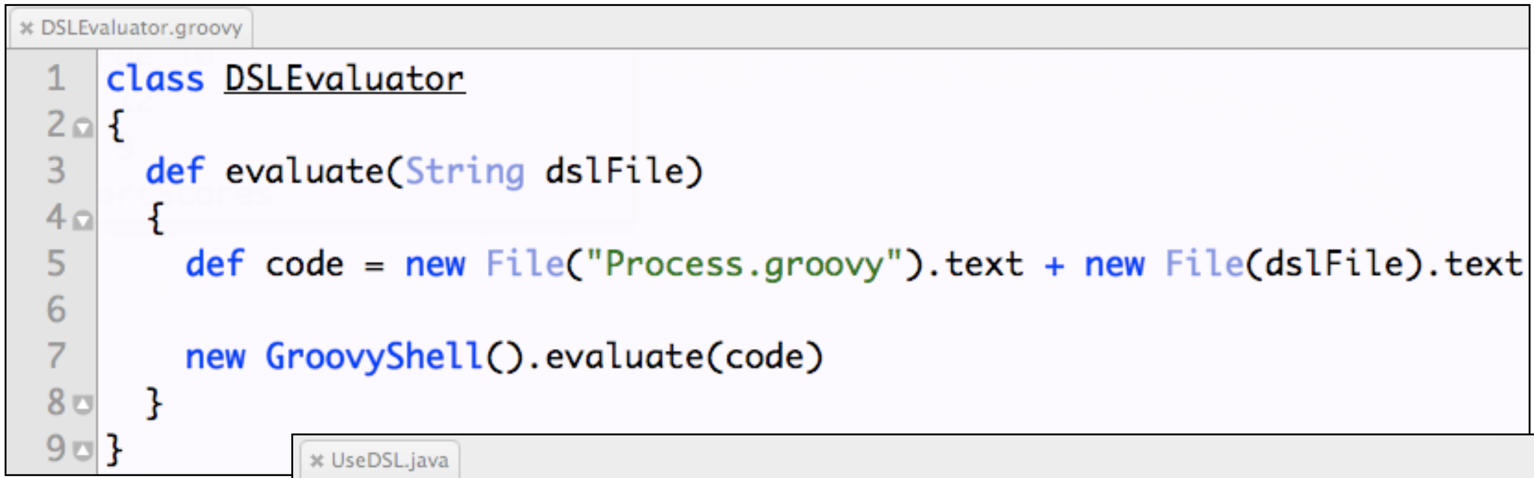


```
> groovyc -j -Jclasspath=$GROOVYCLASSPATH:. UseClass.java GroovyClass.groovy
> java -classpath $GROOVYCLASSPATH:. UseClass
Hello
> echo $GROOVYCLASSPATH
/usr/local/lib/groovy/groovy-1.5.4/embeddable/groovy-all-1.5.4.jar
> 
```

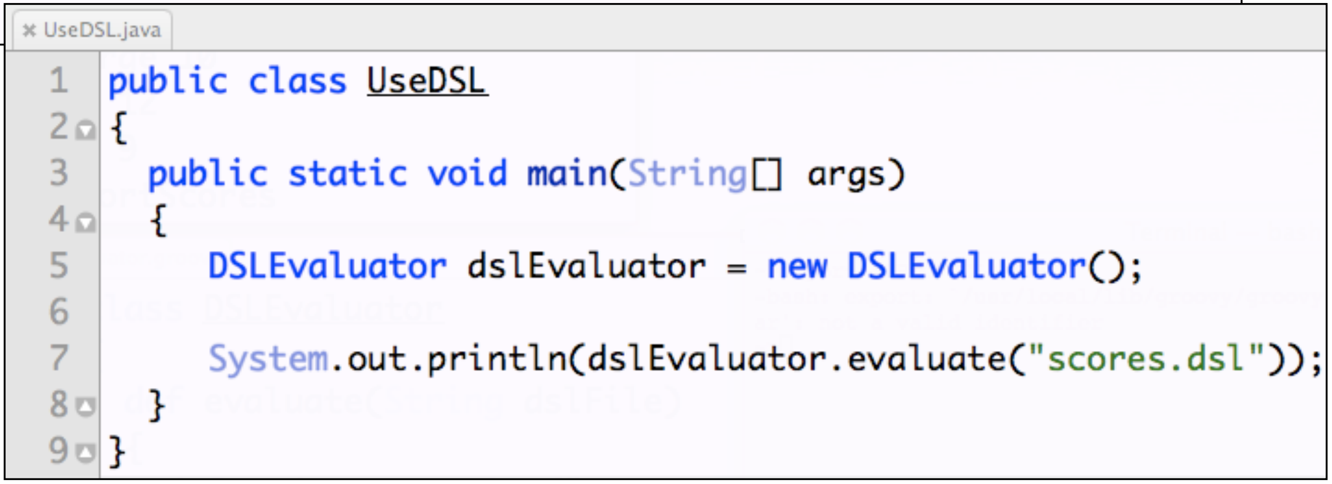
A Small DSL Sample


scores.dsl

```
players 'Ben', 'George', 'Abe'
George 10
Ben 12
Abe 9
reportScores
```

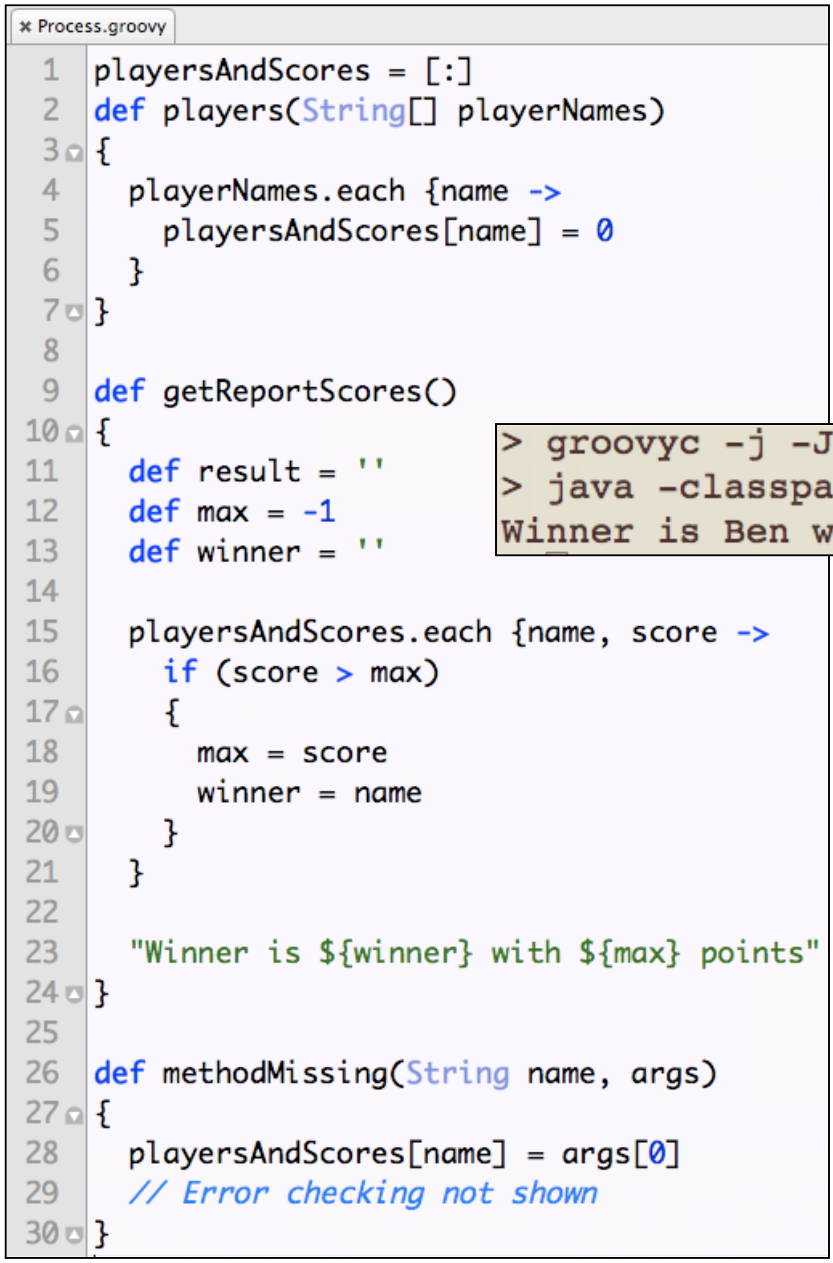

DSLEvaluator.groovy

```
class DSLEvaluator
{
  def evaluate(String dslFile)
  {
    def code = new File("Process.groovy").text + new File(dslFile).text

    new GroovyShell().evaluate(code)
  }
}
```


UseDSL.java

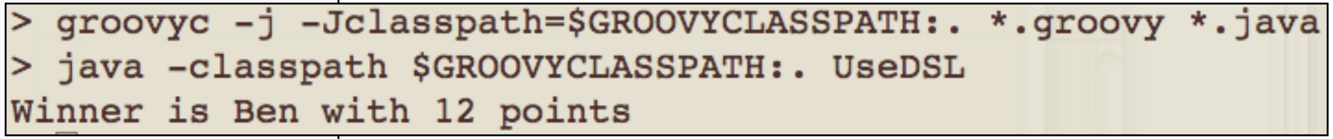
```
public class UseDSL
{
  public static void main(String[] args)
  {
     DSLEvaluator dslEvaluator = new DSLEvaluator();

     System.out.println(dslEvaluator.evaluate("scores.dsl"));
  }
}
```

A Small DSL Sample



Process.groovy

```
playersAndScores = [:]
def players(String[] playerNames)
{
  playerNames.each {name ->
    playersAndScores[name] = 0
  }
}

def getReportScores()
{
  def result = ''
  def max = -1
  def winner = ''

  playersAndScores.each {name, score ->
    if (score > max)
    {
      max = score
      winner = name
    }
  }

  "Winner is ${winner} with ${max} points"
}

def methodMissing(String name, args)
{
  playersAndScores[name] = args[0]
  // Error checking not shown
}
```



```
> groovyc -j -Jclasspath=$GROOVYCLASSPATH:. *.groovy *.java
> java -classpath $GROOVYCLASSPATH:. UseDSL
Winner is Ben with 12 points
```

References

- http://groovy.codehaus.org
- https://scripting.dev.java.net/
- “Programming Groovy: Dynamic Productivity for the Java Developer,” by Venkat Subramaniam, Pragmatic Bookshelf, 2008.

You can download examples and slides from

http://www.agiledeveloper.com - download

Thank You!

Please fill in your session evaluations

You can download examples and slides from

http://www.agiledeveloper.com - download