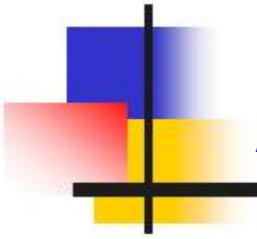


Groovy

A Language Introduction for Java Programmers

Eric Schreiner
Managing Director
Contecon Software GmbH

Other Groovy related sessions

- Eric Schreiner
  - Groovy Modules
    Getting the Power Out of Groovy
- Scott Davis
  - Introduction to Grails

During the presentation: If you have questions – please ask

Prerequisites

- No experience with Groovy is assumed
- Understanding of Java would be helpful.

Conventions used in this presentation

- *Blue italic* will be used for keywords and operators in the text (not in example code)
- All class names for Java classes used in my examples will start with a capital *J* to differ them from Groovy
- Additional example code is indicated in ***green*** at the bottom of the slides. ***Grey*** indicates that we will skip the sample during the presentation (time is limited)

Agenda

- General thoughts
- Setting up the environment
- How to run a script
- Language features
  - Datatypes
  - Collections (Ranges, Lists, Maps)
  - Closures, Flowcontrol, *etc*.
- Questions

Definitions

- ## Wikipedia

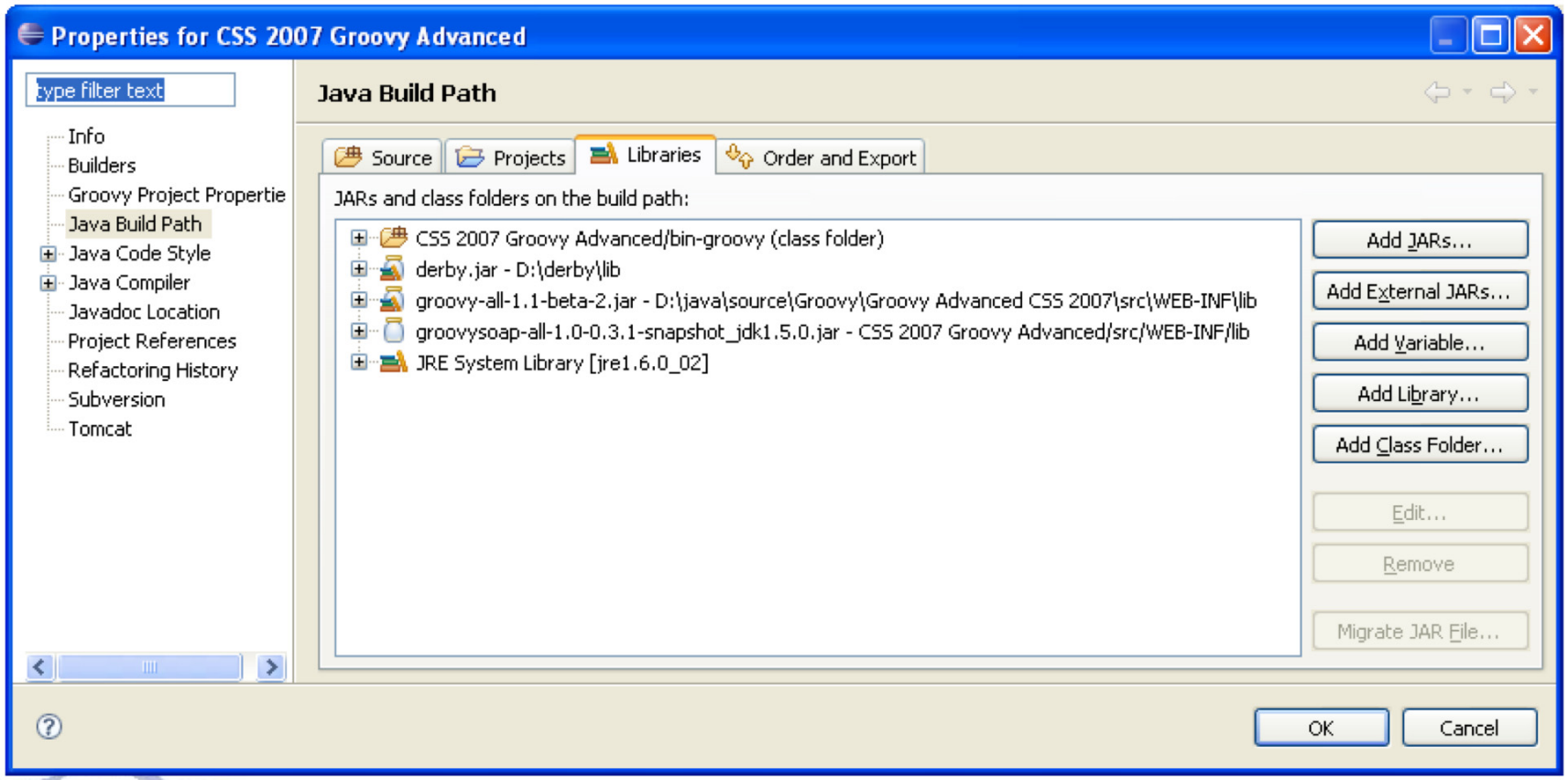
  Groovy is an object-oriented programming language for the Java Platform as an alternative to the Java programming language. It can be viewed as a scripting language for the Java Platform, as it has features similar to those of Python, Ruby, Perl, and Smalltalk. In some contexts, the name JSR 241 is used as an alternate identifier for the Groovy language.

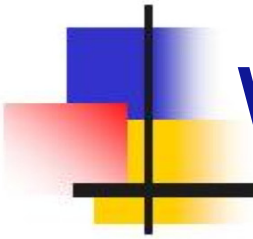
- ## Codehaus.org

  Groovy is like a super version of Java. It can leverage Java's enterprise capabilities but also has cool productivity features like closures, builders and dynamic typing. If you are a developer, tester or script guru, you have to love Groovy.

Eclipse Setup for the examples

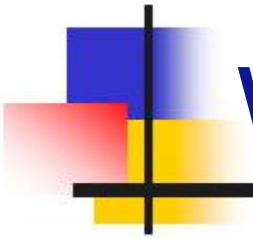
- Plugin: org.codehaus.groovy_1.0.1
  - Compiler output location bin-groovy





Why another language?

- Is Groovy yet another scripting language?
  - YES, but there's more behind it
- Is Groovy better than Python, Ruby, Perl, and Smalltalk?
  - May be
  - I don't know



Why another language?

- Why should I learn Groovy?
  - If you are a Java developer you don't really have to learn it
  - Groovy feels like Java with far fewer restrictions
  - Groovy makes modern programming features available to Java developers with almost-zero learning curve

An experienced Java programmer is more productive with a Java-like scripting language!

Hello World

- Java

```
public class HelloWorld {
 public static void main(String[] args) {
    System.out.println("Hello world");
    }
 }
```

- Groovy

```
println 'Hello World'
```

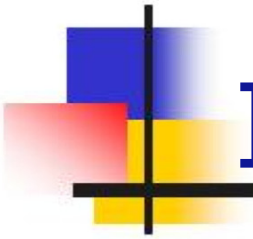
HelloWorld.groovy, JHelloWorld.java

Running Groovy

- At Least JDK 1.4 required
- Install Groovy (see http://www.codehaus.org)
- Commands provided by the Groovy GDK
  - *groovysh* – The Groovy Shell
  - *groovyconsole* – Swing Console
  - *groovy* – Run a Groovy Script
  - *groovyc* – Groovy Compiler
  - *groovyc* – Ant task

Running our Hello Word

- Running with the groovy command
- Use the groovy shell
- Compile Hello world
  - See my other session for more details on what happens under the covers
- Run the compiled Hello world with java
- Use Eclipse to run Hello World



IDE support

- Eclipse plug-in (http://groovy.codehaus.org/Eclipse+Plugin)
  - At least Eclipse 3.1 required
- IntelliJ IDEA plug-in
  - At least IDEA 5.0 build #3378 required
- Utraedit
- Jedit
- other

Comments

- *#!* (shebang) only in first line (for unix shells)
- *//* single line comment
- */* ..... *** multiline Comment
- */*** .... *** Javadoc like comment
  - There is a Groovydoc Api
  - No Groovydoc tool yet

Groovy and Java syntax

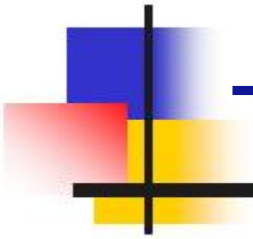
- Groovy is a superset of Java Syntax
  - With some exceptions, like the *for(start,test,inc)* loop or the == operator or the *do{} while* loop
- Most of Java Syntax is also part of groovy
  - Packing mechanism
  - Statements (including package and import)
  - Class and method definition
  - Control structures (see exception above)
  - Operators expressions assignments
  - Object instantiation, referencing, calling methods

Differences to the Java language

- Semicolons and parentheses are optional

```
println 'Hello World'
```

- Automatic import of the following classes and packages
  - java.io.*
  - java.lang.*
  - java.math.BigDecimal
  - java.math.BigInteger
  - java.net.*
  - java.util.*
  - groovy.lang.*
  - groovy.util.*



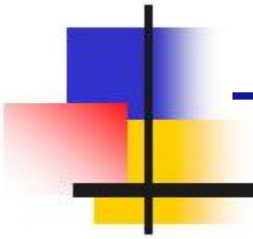
Things to be aware of

- == means always equals
- Use operator *is* to check identity
- *is* is a Keyword
- methods and classes are public by default.
- inner classes are not supported at the moment. (use closures instead)
- All Exceptions are treated like unchecked exceptions

StringEquals.groovy, JStringEquals.java

Features added to the Java language

- Closures
- Native support of Collections
- Support for markup languages like XML, HTML, SAX, W3C DOM, Ant tasks, Swing, *etc*.
- native support for regular expressions
- dynamic and static typing is supported
- Simpler bean support
- safe navigation using the *?.* Operator
  def val = anObject?.aProperty?.aSubProperty



The GDK (Groovy Development Toolkit)

- See (http://groovy.codehaus.org/groovy-jdk.html)
- All Groovy classes are behind the scenes JDK classes
- A lot of useful methods have been added

Type	Java	Groovy
Array	length field	size() method
String	length() method	size() method
StringBuffer	length() method	size() method
Collection	size() method	size() method
Map	size() method	size() method
File	length() method	size() method
Matcher	groupcount method	size() method

FileGDKExamples.groovy

Strings

- Two different types of Strings
  - Plain Strings (instance of *java.lang.String*)
  - GStrings (instance of *groovy.lang.GString*)
    - Allows placeholder expressions
    - Are evaluated at runtime

	Example	Gstring	Backslash escapes
Single quote	'Hi Eric'	No	Yes
Double quote	"hello $name"	Yes	Yes
Tripple single quote	'''============ Total: €10,00 ============'''	No	Yes
Tripple double quote	"""============ Total: €$sum ============"""	Yes	Yes
Forward slash	/x(\d*)y/	Yes	Only \u and \$

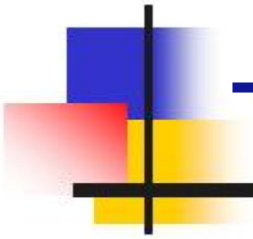
GDK String methods

- A lot of nice methods have been added to Strings
- See http://groovy.codehaus.org/groovy-jdk.html

```
def s="Colorado Software Summit 2007”
assert s.startsWith("Colorado")
assert s[0] == 'C'
assert s[9..16] == 'Software'       // Range
assert s.count("o") == 4
assert s - 'Colorado ' == "Software Summit 2007"

def list = s.tokenize();
assert list.size() == 4
assert list[0] == 'Colorado'
assert list[3] == '2007'
assert list[3].toInteger() == 2007
```

StringGDKSamples.groovy



The groovy type system

- Everything is an Object
- *"def"* is a replacement for a type name.
  In variable definitions it is used to indicate that you don't care about the type
- No primitive types
  - Primitive types can be declared but wrappers are used instead

```
byte    java.lang.Byte
short   java.lang.Short
int     java.lang.Integer
long    java.lang.Long        (literal L or l)
float   java.lang.Float       (literal f or F)
double  java.lang.Double      (literal d or D)
char    java.lang.Character
boolean java.lang.Boolean
        java.math.BigInteger  (literal g or G)
        java.math.BigDecimal  (literal 1.23g or 12.3G)
```

Datatypes.groovy

Operators

Operator	Method	Operator	Method
a + b	a.plus(b)	a == b	a.equals(b)
a - b	a.minus(b)	a != b	! a.equals(b)
a * b	a.multiply(b)	a <=> b	a.compareTo(b)
a / b	a.div(b)	a > b	a.compareTo(b) > 0
a % b	a.mod(b)	a >= b	a.compareTo(b) >= 0
a ** b	a.power(b)	a < b	a.compareTo(b) < 0
a | b	a.or(b)	a <= b	a.compareTo(b) <= 0
a & b	a.and(b)		
a ^ b	a.xor(b)	a as type	a.asType(typeClass)
~a	a.negate()		
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.rightShift(b)		
a >>> b	a.rightShiftUnsigned(b)		

Comparison operators handle nulls gracefully.
So that a == b will never throw a NullPointerException whether a or b or both are null.

OverridingOperator.groovy

Coercion

- When comparing numbers of different types, type coercion rules apply, converting numbers to the largest numeric type before the comparison.

+ - *	B	S	I	C	L	BI	BD	F	D
Byte	I	I	I	I	L	BI	BD	D	D
Short	I	I	I	I	L	BI	BD	D	D
Integer	I	I	I	I	L	BI	BD	D	D
Character	I	I	I	I	L	BI	BD	D	D
Long	L	L	L	L	L	BI	BD	D	D
BigInteger	BI	BI	BI	BI	BI	BI	BD	D	D
BigDecimal	BD	BD	BD	BD	BD	BD	BD	D	D
Float	D	D	D	D	D	D	D	D	D
Double	D	D	D	D	D	D	D	D	D

Coercion1.groovy

Coercion II

- Division
  - Float or Double always results in Double
  - Everything else results in BigDecimal
- Integer division can be achieved by cast or *intdiv()* method
- The shifting operators are only defined for Integer and Long
- The power operator coerces to the next best type in the following sequence:
  Integer, Long, Double

Regular expressions

- Native support of regular expressions
- =~ regex find operator
- ==~ regex match operator (more restrictive)
- ~String regex pattern operator
  this compiles a Java pattern object from a pattern String
- See also
  http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html

Regular expression functions

- Find out if a string matches a pattern
- Find out if there is an occurrence of a pattern in a String
- Count the occurrences of a pattern in a String
- Replace occurrences of all patterns in a String
- Do something with the occurrence
- Split a String by occurrence of a pattern

Collections: Lists

- Native support for Lists
- By default instance of *java.util.ArrayList*
- Can also be created as
  - Sets
  - Other List types
- http://groovy.codehaus.org/Groovy+Collections+In+Depth
- See examples

Collections: Ranges

- Ranges allow you to create a list of sequential values.
- Ranges are Objects
- Range implements *java.util.List*
- Ranges defined with the .. notation are inclusive
- Ranges defined with the ..< notation are exclusive

Collections: Ranges

- Ranges are implemented efficiently, creating a lightweight Java object containing a from and to value
- Ranges can be used for any Java object which implements *java.lang.Comparable* for comparison, and also have methods *next()* and *previous()* to return the next / previous item in the range.

Collections: Maps

- Native support for Maps
- A map item is defined by a key-value pair separated by a *:* (colon)
- By default instance of java.util.HashMap
- Can also be created as
  - TreeMap
  - Other Map types
- http://groovy.codehaus.org/Maps+and+SortedMaps
- See examples

MapSample.groovy

Closures definition

- Wikipedia

  In computer science, a closure is a semantic concept referring to a function paired with an environment. When called, the function can reference elements of the environment required for the function's evaluation....
  The concept of closures was developed in the 60's .....

- For java programmers
  - It's a piece of code wrapped up as a method
  - It's like a Method that can have parameters and a return code without having to declare a Class
  - It's an Object so it can be assigned to a variable

Closures

- Have you ever written code like this?

```
public void rearrangeFromThread() {
    SwingUtilities.invokeLater(
        new Runnable(){
            public void run(){
                panelSomething.invalidate();
                }});
    }
```

anonymous class

- A Groovy version of SwingUtilities.invokeLater with a Closure

```
public void rearrangeFromThread() {
  SwingUtilities.invokeLater(){panelSomething.invalidate();}
  }
```

closure

Question: Can an anonymous class have parameters?

Declaring closures

- { [closureArguments->] statements }
- Example

  { String x, int y -> println "hey ${x} the value is ${y}" }

  - Arguments are optional
  - Closures are always anonymous
  - Closures always return a value. This may occur *via* either an explicit return statement, or as the value of the last statement in the closure body (*e.g.* an explicit return statement is optional).
  - The body of a closure is not executed until it is explicitly invoked (*e.g.* a closure is not invoked at its definition time)
  - Closures are always derived from the class Closure. Code which uses closures may reference them *via* untyped variables or variables typed as Closure.

Closure Parameters

- See http://groovy.codehaus.org/Closures
- Closures may have 1...N arguments, which may be statically typed or untyped. The first parameter is available *via* an implicit untyped argument named ***it*** if no explicit arguments are named. If the caller does not specify any arguments, the first parameter (and, by extension, ***it***) will be null.
- The developer does not have to use ***it*** for the first parameter. If they wish to use a different name, they may specify it in the parameter list.
- A closure may be invoked *via* the call() method, or with a special syntax of an unnamed () invocation. Either invocation will be translated by Groovy into a call to the Closure's doCall() method.

ClosureSample.groovy

Methods as closures

- Referencing a method as a closure is performed using the *reference.&* operator
  - *reference* is used to specify the instance to be used
- Method closures are limited to instance methods
- Runtime parameter overloading is supported (multimethods)

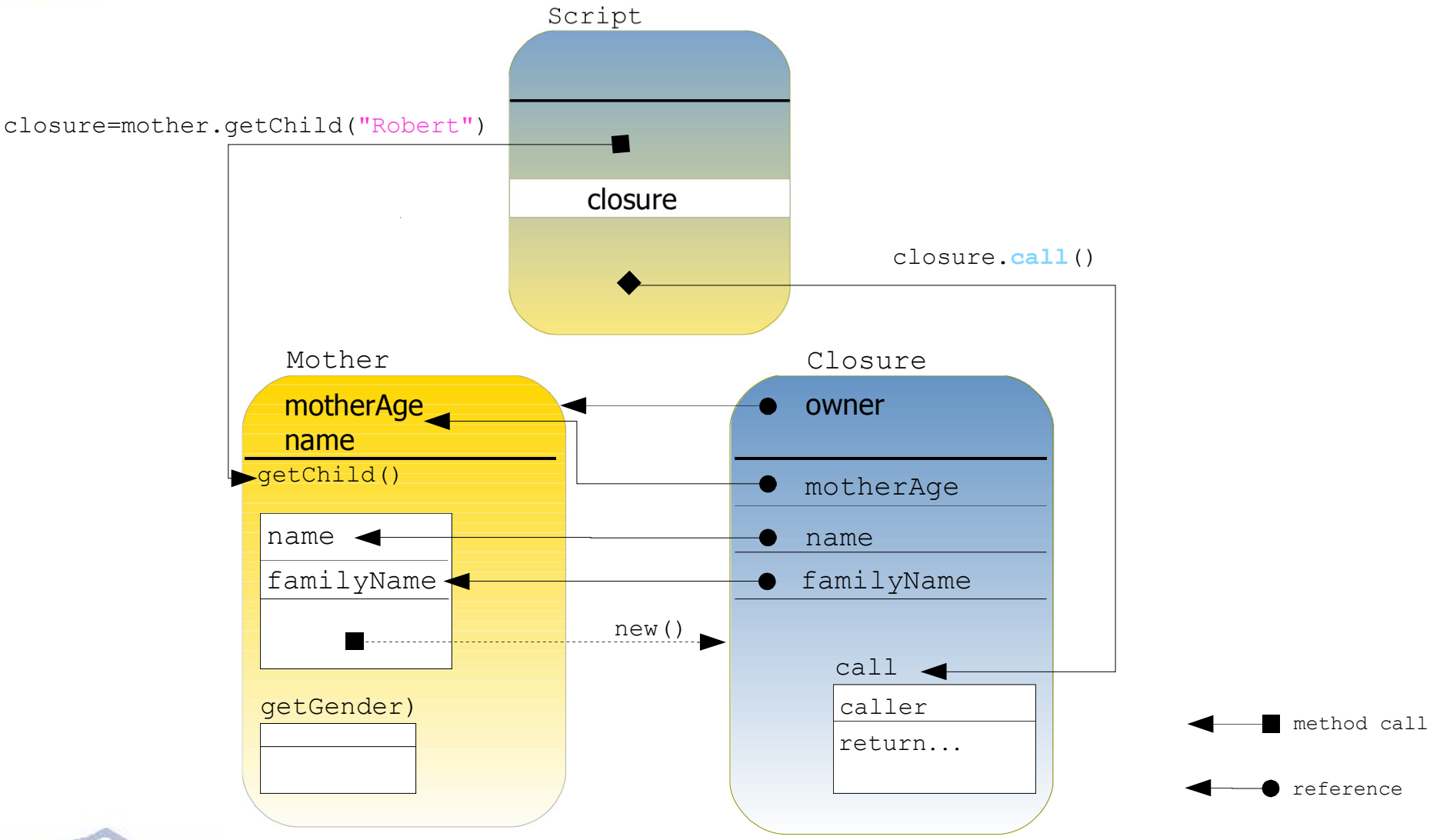
Closures curry and other spices

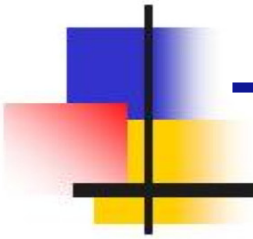
- A closure may be curried so that a copy of the closure is made with one or more of its parameters fixed to a constant value
  see http://en.wikipedia.org/wiki/Currying
- Closures implement the *isCase()* method. The parameter is passed to the closure. The closure method should return a Boolean value

Scoping

- Closures may reference variables external to their own definition
  - local variables
  - method parameters
  - object instance members
- Support of a special *owner* variable to access instance variables of the owning objects
- "*this*" refers to the enclosing class

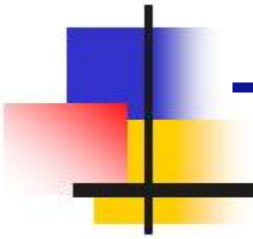
Scoping example





The GDK and Closures

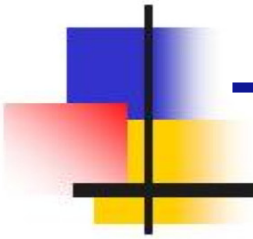
- Object (see http://groovy.codehaus.org/groovy-jdk.html)
  - boolean any(groovy.lang.Closure closure)
    Iterates over every element of a collection, and checks whether a predicate is valid for at least one element
  - boolean every(groovy.lang.Closure closure)
    Iterates over every element of a collection, and checks whether a predicate is valid for all elements.
  - java.lang.Object find(groovy.lang.Closure closure)
    Finds the first value matching the closure condition (returns: java.lang.Object)
  - java.util.List findAll(groovy.lang.Closure closure)
    Finds all values matching the closure condition (returns: java.util.List)
  - int findIndexOf(groovy.lang.Closure closure)
    Iterates over every element of the collection and returns the index of the first object that matches the condition specified in the closure (returns: int)
  - java.util.List grep(java.lang.Object filter)
    Iterates over every element of the collection and returns each object that matches the given filter - calling the isCase() method used by switch statements. This method can be used with different kinds of filters like regular expresions, classes, ranges, *etc*.
    (returns: java.util.List)



The GDK and Closures

- Object (see http://groovy.codehaus.org/groovy-jdk.html)
  - java.util.List collect(groovy.lang.Closure closure)
    java.util.Collection collect(java.util.Collection collection, groovy.lang.Closure closure)
    Iterates through this object transforming each object into a new value using the closure as a transformer, returning a list of transformed values.
  - void each(groovy.lang.Closure closure)
    Allows objects to be iterated through using a closure
  - void eachWithIndex(groovy.lang.Closure closure)
    Allows object to be iterated through a closure with a counter
- Number
  - void step(java.lang.Number to, java.lang.Number stepNumber, groovy.lang.Closure closure)
    Iterates from this number up to the given number using a step increment
  - void times(groovy.lang.Closure closure)
    Iterates a number of times
  - void upto(java.lang.Number to, groovy.lang.Closure closure)
    Iterates from this number up to the given number
  - void downto(java.lang.Number to, groovy.lang.Closure closure)
    Iterates from this number down to the given number

ClosuresGDKSample.groovy



The GDK and Closures

- String
  - eachMatch(java.lang.String regex, groovy.lang.Closure closure)
    Process each regex group matched substring of the given string. If the closure parameter takes one argument, an array with all match groups is passed to it. If the closure takes as many arguments as there are match groups, then each parameter will be one match group.
  - java.lang.String replaceAll(java.lang.String regex, groovy.lang.Closure closure)
    Replaces all occurrencies of a captured group by the result of a closure on that text.
- File (see http://groovy.codehaus.org/groovy-jdk.html)
  - A lot of interesting things like

```
void    eachDir(groovy.lang.Closure closure)
void    eachDirMatch(java.lang.Object filter, groovy.lang.Closure closure)
void    eachDirRecurse(groovy.lang.Closure closure)
void    eachFile(groovy.lang.Closure closure)
void    eachFileMatch(java.lang.Object filter, groovy.lang.Closure closure)
void    eachFileRecurse(groovy.lang.Closure closure)
void    eachLine(groovy.lang.Closure closure)
void    eachObject
void    eachDirRecurse(groovy.lang.Closure closure)
void    eachFile(groovy.lang.Closure closure)
void    eachFileMatch(java.lang.Object filter, groovy.lang.Closure closure)
void    eachFileRecurse(groovy.lang.Closure closure)
void    eachLine(groovy.lang.Closure closure)
void    eachObject
```

Control structures

- Boolean testing

Type	Evaluation condition for true
Boolean	Value is true
Matcher	The matcher has a match
Collection	Not empty
Map	Not empty
String	Not empty
Gstring	Not empty
Number	The value is nonzero
Character	The value is nonzero
None of the above	The object reference is not null

Assignments and boolean tests

- Assignments and testing of none boolean expression is not allowed in *if* Statements (a subexpression must be used instead)

```
if((i = 3)) {
  println i
  assert i == 3
  }
```

- It is allowed for *do* and *while* statements

```
while(i=i-1) {
     println i
     }
```

Conditional execution

- *if* and *else* is like in Java
- Also the *:?* (ternary) operator
- The switch statement adds a lot of functionality
  - Classifiers – by using isCase()
  - Type case
  - Closure case
  - Regular expression case

Switch with classifiers

```
switch(candidate) {
      case classifier1:  doSomething1(); break;
      case classifier2:  doSomething2(); break;
      default:           doNothing();
      }
```

```
if      (classifier1.isCase(candidate)) doSomething1()
else if(classifier2.isCase(candidate)) doSomething2()
else                                   doNothing();
```

Implementation of isCase()	
Class	a.isCase(b) implementation
Object	a.equals(b)
Class	a.isInstance(b)
Collection	a.contains(b)
Range	a.contains(b)
Pattern	a.matcher(b.toString()).matches()
String	(a==null && b==null) || a.equals(b)
Closure	a.call(b)

SwitchSample.groovy

Looping

- *while*
  - Like in Java
    with extended boolean expressions
- *do {} while*
  does not exist
- *for(start,test,inc)*
  does not exist

Other control Structures

- Exiting blocks
  - *return*: like in Java with the following exceptions
    - If omitted the result of the last expression will be returned
    - Methods with return type void do not return a value
    - Closures always return a value (null if last expression was a call to a void method)
  - *break*: like in Java
  - *continue*: like in Java
- Exceptions
  - like in Java - with the difference that the declaration of exceptions in the method signature is optional – even for checked exceptions
  - Braces are always required around the try-catch block bodies

Scripts and Classes

- Class definition is almost identical to Java
  - A class may contain fields, constructors, initializers and methods
  - Methods and constructors may contain local variables
- Like in Java files may be organized in packages
- Classpath
  Groovy checks **.class* and **.groovy* files. When the classloader finds both .class and .groovy it uses whichever is newer

Scripts and Classes

- File contains no class definition
  - A class will be generated that extends *Script*
- File contains exactly one class definition
  - Same as in Java
- File contains multiple class definitions
  - Compiler creates multiple .class files
- File contains scripting code and class definition
  - A main method is generated for the script

VariableDeclarationSample.groovy JAD(www.kpdus.com/jad.html) HelloWorld.groovy

Fields

- If no visibility is specified a Property will be generated
  - Groovy Beans -> Java Beans
  - Accessor methods will be generated
- Fields / properties may be accessed by
  - By using accessors
  - By using the subscript operator
  - By using the field access operator

Expandos

- Expandable alternative to a bean
- Supports Groovy style property access
- It's possible to assign a closure to a field

Methods

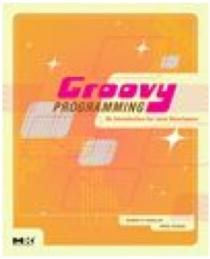
- Java modifiers can be used
- Default visibility is public
- Return type is optional
  - *java.lang.object* will be returned if type not specified
  - Use def if no return type or visibility is available
- Arguments with default values

Constructors

- If no constructor is given an implicit constructor will be created
- Constructors are public by default
- Constructors may be called in three different ways
  - The usual Java way
  - With positional parameters
  - With named parameters

References

- Groovy
  - Codehaus
    http://groovy.codehaus.org/
  - JSR241
    http://www.jcp.org/en/jsr/detail?id=241
  - developerWorks
    http://www-128.ibm.com/developerworks/java/library/j-alj08034.h
- Books
  - Groovy in Action (very good)
  - Groovy Programming



Contact me

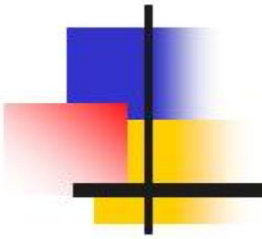
Eric Schreiner

Eric.Schreiner@contecon.de

http://www.contecon.de

Sources: http://www.contecon.de/groovy/

Please fill out the evaluations



Thank you

....if you have questions

ask now