Introduction to Groovy

Part I – Compare & Contrast with Java

About the Labs

The labs are downloadable source code files packaged in a zip file. The download link will be provided at the workshop.

Instructions for each lab are in comments in the source file. If you find a TODO, there's workto do!

The names of the labs are like LabX_NNN, where X corresponds to the part of this workbook and NNN is in the range of 100-499

- The 100s labs are very simple "Uncomment / type this and run to see the results" kind of labs
- The 200s labs are more complex, but still on the easier level with many hints
- The 300s labs may introduce new concepts not in the materials and few hints on the basics
- The 400s labs are almost purely functional with little extra coding guidance available; i.e. "Given a list of Orders (where the Order class is defined and some sample data provided), find all orders in USD with taxable amounts more than $7250 and nontaxable amounts less than $1750."

Overview

- About Groovy
- Getting Started
- Groovy Scripts & Classes
- POGO v POJO
- Groovy Strings
- Dynamic Typing
- A Taste of Dynamic Behavior
- Some Gotchas
- What's Next

About Groovy

A Brief History…

2003

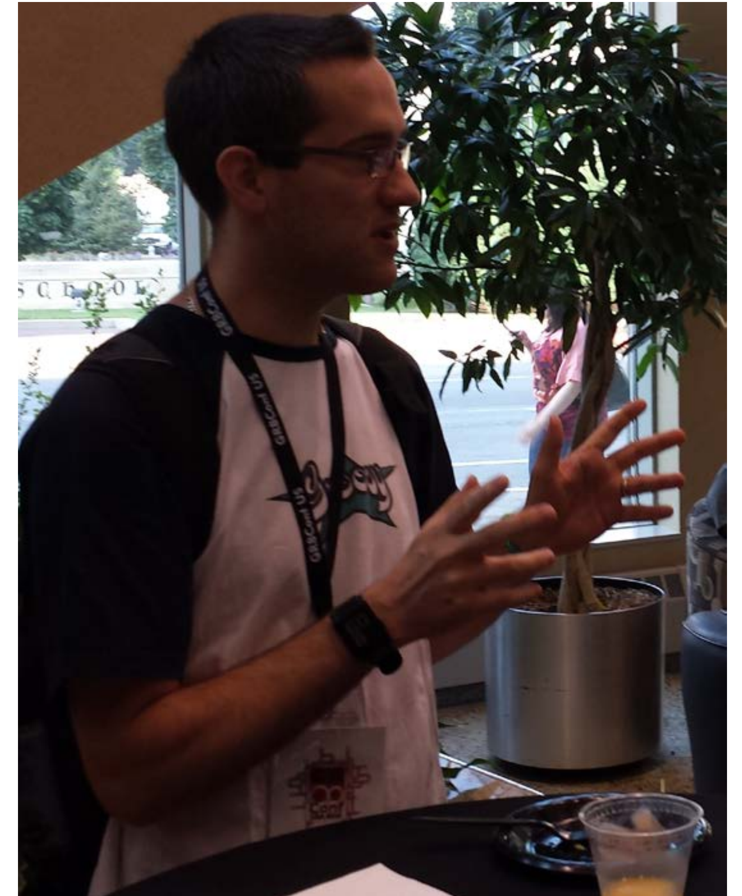
- James Strachan announces the birth of Groovy as a project
- Java language look-and-feel
- Dynamic features modeled from Ruby, Python
- Guillame LaForge starts exploring Groovy

2004

- Groovy submitted to JCP (JSR 241)

2006

- James Strachan moves on from the Groovy project
- Guillame LaForge takes over



About Groovy

A Brief History…

2007

- Groovy 1.0 released after several betas
- G2One company is formed by Graeme Rocher (creator of Grails)
- LaForge is made Groovy Project Lead
- Groovy 1.0 and 1.1 released, then rebranded to 1.5 to reflect rapid changes

2008

- SpringSource acquires G2One

2009

- VMware acquires SpringSource

About Groovy

A Brief History…

2012

- Slow JCP process is abandoned and LaForge lists JSR 241 as dormant
- Groovy 2.0 released

2013

- VMware spins off SpringSource to Pivotal Software, Inc.

2014

- Groovy v2.3 released
- Current version is 2.3.6
  - Includes traits, Java 8 support, and more

About Groovy

So what *exactly* is Groovy?

- Groovy is a dynamic language based on Java
- Groovy objects *are* java.lang.Objects
- Groovy objects implement groovy.lang.GroovyObject interface
- Groovy GDK defines Groovy-specific types
- Groovy also dynamically *enhances* artifacts in the JDK

Getting Started

For this talk, we'll get Groovy manually. We'll discuss including Groovy as a build dependency in Part II

1. Download zip distribution from groovy.codehaus.org/Download
2. Unzip to install it
3. Set GROOVY_HOME
4. Set JAVA_HOME
5. Begin the Groovy love

Groovy Scripts and Classes

```
// file: Hello.groovy
// Groovy code can be in a class like Java
class Hello {
  static void main(args) {
    println “Hello World!”
  }
}
```

```
// file: SayHello.script
// Groovy code can be in a script
println “Hello World!”
```

POGO v POJO

Let's start with a Java POJO and make it into a Groovy POGO…

```
// a POJO

import java.util.Date;

public class Member {
    private int memberId;
    private String screenName;
    private String firstName;
    private String lastName;
    private String email;
    private Date joinDate;

 public int getMemberId() { return memberId; }
    public void setMemberId(int memberId){this.memberId = memberId;}

    public String getScreenName() { return screenName; }
    public void setScreenName(String screenName) {this.screenName =
                                              screenName; }

    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }

    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public Date getJoinDate() { return joinDate; }
    public void setJoinDate(Date joinDate) { this.joinDate = joinDate; }

    @Override
    public String toString() {
        return String.format("Member(%d, %s, %s, %s, %s, %s)",
                             memberId, screenName, firstName, lastName, email,
                             joinDate.toString());
    }
}
```

POGO v POJO

Let’s start with a Java POJO and make it into a Groovy POGO…

```
// POGO

@groovy.transform.ToString
class Member {
    int memberId
    String screenName
    String firstName
    String lastName
    String email
    Date joinDate
}
```

Groovy Strings

Let’s explore the four ways we can make String literals…

- Single quote

  ‘Hello World!’

- Double Quote

  “Hello World” – a java.lang.String

  “Hello $name” – string interpolation token ($name) makes this a groovy.lang.Gstring

- Triple Single Quote Heredoc

```
“’<airlines>
    <airline>American</airline>
    <airline>Southwest</airline>
    <airline>Virgin America</airline>
  </airlines>’’’
```

- Triple Double Quote Heredoc

```
“””<airlines>
    <airline>${airlines[0]}</airline>
    <airline>${airlines[1]}</airline>
    <airline>${airlines[2]}</airline>
  </airlines>”””
```

Dynamic Typing

Duck Typing…

If it walks like a duck and quacks like a duck, let's treat it like a duck…

List list = [3, 2, 4, 1, 0, 5]

list.sort() // yields [0,1,2,3,4,5] … but where did sort() come from?

Groovy compiler didn't complain…

It assumes sort() will be there are runtime

Dynamic Behaviors

With great power, comes great responsibility…

Delegate means ‘the thing that should get this behavior’; i.e. the String

String.metaClass.asCanadian = { delegate + ‘, eh’ }

println ‘Nice weather’.asCanadian() // yields ‘Nice weather, eh’

Some Gotchas

You can almost rename a .java source file to .groovy, but not quite…

In Groovy:

- a == b means a.equals(b)
- By default, no way to declare package-private visibility
- do/while not supported
- Floating point literals are BigDecimals, not doubles
- Array literal declaration won't work: int[] x = {1,2,3}. Use int[] x = [1,2,3]
- Groovy compiler will not check throws clause on method as all exceptions treated the same way by the compiler

Part I Self-Examination

1. In what year was Groovy announced?
2. What GA release of Groovy first introduced Traits?
3. Groovy is a strongly typed statically compiled language. (T/F)
4. Groovy objects are java.lang.Objects. (T/F)
5. All Groovy objects implement what interface?
6. All code you write in Groovy must be inside a class. (T/F)
7. The ‘A’ literal specified with single quotes is what type?
8. The “A-$code” literal is an instanceof groovy.lang.GString. (T/F)
9. A triple quoted literal is called a h_ _ _ d _ _
10. A method call to a non-existent method results in a compiler error. (T/F)
11. Given a1 == a2 compare whether a1 and a2 object references are the same. (T/F)

Part I Labs

Please complete the Lab1_1xx labs

Time permitting, you may go on to the Lab1_2xx, 3xx, and 4xx labs.

However, please complete the easier labs before turning to the more challenging ones as the later labs build on concepts practiced in the earlier labs.

Introduction to Groovy

Part II – Groovy Grooviness

Overview

Syntactical Sugar

Groovy truth

Groovy operators

Closures, A First Look

Intro to Collections

First, A Tale of Two Types of Complexity…

Essential complexity

The problems we try to solve have essential complexity. By their nature, they are complex

Accidental complexity

All the complexity we add in to implement a solution to the problem that is not essential complexity is accidental complexity

Using Groovy’s syntactical sugars and idioms, we can drive out a lot of the accidental complexity we added in because of Java

Syntactical Sugar

More default imported packages than Java

All classes assumed public

All fields assumed private; public getters / setters added automatically

Free map c'tor

Access / mutate properties like field level access (but it's not)

Parentheses optional in method calls (unless no arg)

Semi-colons optional unless multiple statements on line (or classic for loop)

Return statement optional at end of methods. Last evaluated expression is returned. Don't be obtuse!

Void methods return null

The Groovy compiler treats all exceptions as Runtime

Groovy Truth

Java is very rigid about truth

- if(someExpression) {…} // someExpression must be boolean result

This leads to a lot of noise in the code

```
if(arry != null && arry.length > 0)    // arrays have length property
if(name != null && name.length() > 0) // Strings have length()
if(list != null && list.size() > 0)    // collections have size()
if(map != null && map.size() > 0)      // maps have size()
if(iter != null & iter.hasNext())      // iterators
if(value != 0)                         // number not zero is true
if(anyObject != null)                  // object not null is true
```

Groovy Truth

Groovy has a relaxed definition of truth

- It's like JavaScript's definition

This leads to a lot of noise in the code

```
if(anyObject)      // true if not null
if(arry)           // true if array is not null and size() > 0
if(name)           // true if String is not null and size() > 0
if(list)           // true if collection not null and size() > 0
if(map)            // true if map is not null and size() > 0
if(iter)           // true if iteration (or enumeration) not null & has more
if(value)          // true if numeric not null and not zero

/* Groovy adds size() to arrays and Strings so we can finally have a
   consistent syntax about a thing's size */
```

Groovy Operators

Safe Navigation (?.)

Elvis (?:)

Spaceship (compareTo) (<=>)

Spread (*.)

Range operator (..)

getAt/putAt ([])

asType (as)

Regex find (=~) / Regex match (==~)

Method reference (.&)

Membership (in)

Identity (is)

Closures, A First Look

Groovy closures are first class functional citizens

Think of them as disembodied methods that can be passed around

```
Closure max = { a, b -> a > b ? a : b }
```

There's an elegant syntax for declaring and passing them inline

- If a method takes a Closure as its last parameter, the closure can be declared inline
- For example, Groovy associates a collect method with java.util.Collection that takes a transforming closure as the (last) argument allowing it to be called two ways:

```
Closure square = { it * it }         // arg name defaults to 'it'
[1,2,3].collect(square)              // yields [1,4,9]
[4,5,6].collect{val -> val * val } // yields [16,25,36]
```

Intro to collections in Groovy

Groovy adds many enhancements to JDK collections

- collect, find, findAll, sort and many more

Groovy introduces a new type called a Range

```
Range r = 1..5
r.each{ print "$it " }  // yields: 1 2 3 4 5
```

Lists feel more like arrays

```
def vals = [1,2,3,4,5]  // a java.util.ArrayList
println vals[2]         // 3
println vals[-1]        // 5
```

Maps feel more like beans. Simple string keys need not have quotes.

```
def vals = [a:1,b:2,c:3] // a java.util.LinkedHashMap
println vals['a']        // 1
println vals.c           // 3
```

Part II Self-Examination

1. Java overrides the + operator for Strings, but not BigDecimals. Adding BigDecimals a and b requires us to use a.add(b) instead of a+b. This is an example of ________ complexity imposed by the language.
2. The following are equivalent: println('Hello') println 'Hello' (T/F)
3. If a method takes no arguments, parentheses are optional when calling the method (T/F)
4. When returning a value from a method the return statement is always required (T/F)
5. In Groovy truth, a non-null, but empty list is evaluated as true. (T/F)
6. In Groovy truth, a non-zero Integer is evaluated as true. (T/F)
7. Using the ___ Groovy operator we can shorten this common Java idiom: **`String x = name != null ? name : "Anon"`**
8. Closures are behaviors that are first class types that can be passed around in the system. (T/F)
9. The following declares a _____: def x = [1,3,5,7]
10. The following declares a _____: def y = 1..99
11. The following declares a _____: def z = [a:2, b:4, c: 6]

Part II Labs

Please complete the Lab2_1xx labs

Time permitting, you may go on to the Lab2_2xx, 3xx, and 4xx labs.

However, please complete the easier labs before turning to the more challenging ones as the later labs build on concepts practiced in the earlier labs.

Introduction to Groovy

Part III – More Groovy Grooviness

Overview

More on Groovy SDK

Traits

More on Groovy SDK

Groovy Operator Overloading

More java.util Collections JDK enhancements

java.lang.String JDK enhancements

java.lang.Integer JDK enhancements

Groovy Operator Overloading

Groovy provides an elegant way to override many operators

Each operator that can be overloaded has a corresponding method named like the operator

- + is overloaded with plus()
- - is overloaded with minus()
- ++ is overloaded with next()
- -- is overloaded with previous()
- << is overloaded with leftShift()
- >> is overloaded with rightShift()
- [] is overloaded with getAt and putAt; i.e. a[b] and a[b]=c
- Many more!

Groovy Operator Overloading

All we have to do is implement the operator method in our class to give meaning to the operator

To support chaining, the return from the method is the reference to the object with the operator behavior

Groovy Operator Overloading

```
@groovy.transform.ToString
class Pizza {
  String name
  List<String> toppings = []
  Pizza leftShift(String topping) {
    toppings.add(topping)
    this
  }
}
Pizza p = new Pizza(name: "Simply Yummy")
p << 'Cheese' << 'Mushroom' << 'Pepperoni' << 'Sausage'
```

Groovy Operator Overloading

Some examples from the Groovy SDK

- All collections have +, -, and <<
- Number implements +, -, *, / and ** (power) operators
  - BigDecimals get these behaviors
- Dates
  - Have + and – support, as well as ++ and -- for adding/subtracting days from a date
  - Have a getAt and putAt method for getting / setting fields

More on collections enhancements

Spread operator (*) works across elements of a collection

```
List list = [‘My’, ‘name’, ‘is’, ‘Jack’]
assert list.size() == 4
assert list*.size() == [2, 4, 2, 4]
```

More on collections enhancements

Given this List:

List list = [1,2,3,4,5]

We can reverse the list:

list.reverse() == [5,4,3,2,1]

We can join the list elements

list.join('-') == '1-2-3-4-5'

We can find the first and last elements

list.first() == 1

list.last() == 5

We can even find all permutations of a list

list.permutations == [[1, 2, 4, 5, 3], [5, 3, 2, 4, 1], [1, 3, 2, 5, 4], [3, 1, 2, 4, 5], [3, 4, 2, 5, 1],,…]

More on collections enhancements

Given this collection:

```
List list = [‘My’, ‘name’, ‘is’, ‘Jack’]
```

Collect method takes a closure and applies it to each element yielding a transformed element:

```
List newList = list.collect { it.toUpperCase() }
assert newList == [‘MY’, ‘NAME’, ‘IS’, ‘JACK’]
```

More on collections enhancements

Given this collection:

```
List list = [‘My’, ‘name’, ‘is’, ‘Jack’]
```

We can slice into the list using subscripts and ranges

```
assert list[3] == [‘Jack’]
assert list[1..3] == [‘name’, ‘is’, ‘Jack’]
```

More on collections enhancements

Collections also have

- find(Closure) – find first element matching condition in closure
- findAll(Closure) - find all elements matching condition in closure
- inject method taking a closure with two arguments, the initial value and the element value

Combining these, we can perform a Filter/Map/Reduce operation with findAll, collect and inject methods

First, let’s see this in Java

More on collections enhancements

```
// LineItem has boolean taxable, int qty and BigDecimal price
public BigDecimal calculateTaxableTotal(List<LineItem> items) {
  BigDecimal sum = BigDecimal.ZERO;
  for(LineItem item: items) {
    if(item.isTaxable()) {
      sum = sum.add(item.getPrice().multiply(item.getQty()));
    }
  }
  return sum;
}
```

More on collections enhancements

```
// LineItem has boolean taxable, int qty and BigDecimal price
BigDecimal calculateTaxableTotal(List<LineItem> items) {
  items.findAll{ it.taxable }
       .collect{ it.price * it.qty }
       .inject { sum, cost -> sum += cost }
}
```

Some String enhancements

We’ve seen String interpolation

```
“Hello $name”
```

And Heredocs:

```
'''
<customer>
  <name>Acme></name>
  <id>1234></id>
</customer>
'''
```

Some String enhancements

There are more … a lot more!

```
capitalize : ‘hello’.capitalize() == ‘Hello’
reverse: ‘hello’.reverse() == ‘olleh’
isXXX: ‘1.0’.isInteger() == false
       ‘1.0’.isDouble() == true
execute: ‘cmd.exe /c dir’.execute() // lists dir
center: ‘ banner ’.center(20, ‘*’)
           == ‘****** banner ******’
padLeft/Right:  '123'.padLeft(6, '_') == ‘___123’
                '1234'.padLeft(6, '_') == ‘__1234’
```

Some Integer Enhancements

```
times: 10.times{ print “$it “ } // prints 0 1 2 3 4 5 6 7 8 9

power: 2**16 == 65536

upto: 1.upto(10) { print "$it " } // prints 1 2 3 4 5 6 7 8 9 10
```

Traits

Traits are new in Groovy 2.3

Declared like an interface or class, except with the ***trait*** keyword

```
trait Sailing {
  void sail() { println “I’m sailing!” }
}
```

They kind of feel like classes

- TraitA can extend one trait and implement zero or more interfaces
- TraitA can also implement multiple traits: `trait TraitA implements TraitB, TraitC, TraitD`
- They can have state
- They can declare abstract methods

However, they’re used by classes like interfaces

- ClassA implements TraitA, TraitB

Traits

Traits are new in Groovy 2.3

```
trait Flying {
  String airplaneType
  void fly() { println “I’m flying a $airplaneType!”
}
trait Sailing {
  void sail() { println “I’m sailing!” }
}
class Person implements Flying, Sailing {}


Person p = new Person(airplaneType: ‘Boeing 737’)
p.fly()  // prints I’m flying a Boeing 737
p.sail() // prints I’m sailing!
```

Traits

What if there's a method clash? Last declared implements, wins

```
trait SailingA {
  void sail() { println "I'm sailing A!" }
}
trait SailingB {
  void sail() { println "I'm sailing B!" }
}

class Person implements SailingA, SailingB {}

Person p = new Person()
p.sail() // prints I'm sailing B
// We can override this ordering by overriding method in Person
// and calling SailingA.super.sail()
```

Traits

Classes can override trait methods

```
trait Boating {
  void sail() { println “I’m sailing!” }
  void row() { println “I’m rowing!” }
}

class Person implements Boating {
  void sail() { println “I’m sailboating!” }
}


Person p = new Person()
p.sail() // prints from class: I’m sailboating!
p.row() // prints from trait: I’m rowing!
```

Traits

In Groovy (not Java), we can implement traits at runtime rather than compile time

```
trait Named {
  String name
  void sayName() {
    println "Hello $name"
  }
}
trait Aged {
  String age
  void sayAge() {
    println "Age: $age"
  }
}
```

```
class Person {}

def p1 = new Person() as Named
p1.name = 'Jack'
p1.sayName() // prints Hello Jack

def p2 = p1.withTraits Named, Aged
p2.name = 'Jill'
p2.age = 29
p2.sayName() // prints Hello Jack
p2.sayAge()  // prints Age: 54
```

Traits

A few other points

- Traits are compiled into the code
- Compatible with @CompileStatic
- Traits can be chained together (think Servlet Filters) to pass processing through to implemented traits earlier in the chain
- A method on TraitC can call super.x() to pass control to prior trait, TraitB
- If TraitB doesn't have x(), super search will continue to TraitA, then the Bar class

```
class Foo extends Bar implements TraitA, TraitB, TraitC
```

Part III Self-Examination

1. To overload the + operator on a class, simply implement a method named ____
2. What is the result of this: `[‘Santa’,‘Claus’,‘Knows’]*.size()`
3. What is the result of this: `[‘A’,‘B’,‘C’].join(‘,’)`
4. What is the result of this: `2**3`
5. What is the result of this: `‘ banner ’.center(5, ‘*’)`
6. As of Groovy v2.3, ***trait*** is a new keyword (T/F)
7. Do classes extend or implement a trait?
8. Like interfaces, traits can extend multiple traits. (T/F)
9. Like a class, traits can implement interfaces (T/F)
10. Traits can implement other traits (T/F)

Part III Labs

Please complete the Lab3_1xx labs

Time permitting, you may go on to the Lab3_2xx, 3xx, and 4xx labs.

However, please complete the easier labs before turning to the more challenging ones as the later labs build on concepts practiced in the earlier labs.

Introduction to Groovy

Part IV – More Groovy Basics

Overview

Some useful AST transformations

Understanding the Groovy Meta-Object Protocol (MOP)

Builders and Slurpers

Adding Groovy to your Maven Projects

Using Groovy with Your Favorite IDEs

Some useful AST transformations

First, what the heck is a Groovy AST Transformation?

- AST == Abstract Syntax Tree
- When code is compiled, it is broken down into a hierarchical graph of syntax elements
- AST Transformations participate in the compiler's processing to alter the compiled output
  - This means , unlike dynamic behaviors, Java can see the ASTx code!

In Groovy, an AST Transformation is represented by an annotation

Writing AST Transformations is a complex business beyond the scope of this talk

- There are online and book resources that discuss the process
- The process will soon be easier thanks to some tools and DSLs in early stages of development

AST Transformations in the Groovy SDK are in groovy.transform

Some useful AST transformations

@ToString

- Adds a toString method
- Takes optional arguments to
  - **includeNames** – include field names
  - **includeFields** – include private attributes in addition to properties
  - **includePackage** - include package names of properties/fields
  - **includes/excludes** – include or exclude specific fields and properties by name. Use one or the other, but not both
  - **includeSuper** – whether to include super fields/properties
  - **ignoreNulls** – don't display fields or properties with null values
  - **cache** – whether to cache toString results

Some useful AST transformations

@TupleConstructor

- Adds a tuple-style constructor with parameters for each field / property
  - Parms are in the order the fields are declared
  - If includeSuperProperties is set, parms for the the super fields appear first
- Default values (the Java defaults) are provided for each argument so you can leave off any number from the end
  - This provides ability for c'tor to be used as a default no-arg c'tor
  - Also, Groovy's map c'tor is usually available. See GroovyDoc for limitations
- Takes optional arguments to
  - **callSuper** passes args in super call rather than setting properties
  - **includes/excludes** - allows specifying fields and/or properties by name to include or exclude. Use one or the other, but not both
  - **includeFields/includeProperties** – include fields / include properties in c'tor
  - **includeSuperFields/includeSuperProperties** – include super attributes in c'tor
  - **force** overrides suppression of generated c'tor if custom c'tors present

Some useful AST transformations

@EqualsAndHashCode

- Adds an equals and hashCode method
- Takes optional arguments to
  - **callSuper** – whether to include super
  - **includes/excludes** – allows specifying fields and/or properties by name to include or exclude. Use one or the other, but not both
  - **includeFields/includeProperties** – include fields / include properties in c'tor
  - **useCanEqual** – Generates a canEqual method to be used by equals. Default is true. See GroovyDocs for details on this
  - **cache** – whether to cache hashCode calculations

Some useful AST transformations

What would we get in the .class if we create a class like this?

```
@TupleConstructor
@EqualsAndHashCode
@ToString
class Person {
  String firstName
  String lastName
  String email
}
```

That's nice, but do I really have to repeat those three ASTx? Nope.

Some useful AST transformations

@Canonical

- Equivalent to @TupleConstructor, @EqualsAndHashCode, @ToString
- However, it's more limited in options
- Adds a default c'tor
- Adds a tuple-style c'tor taking fields in the order they are declared
  - Map c'tor may not be available. See GroovyDocs
- Adds default equals, hashCode, and toString methods
- Note: C'tors added only if you don't write one of your own
- Other more specific AST Transformations take precedence; i.e. @ToString
- Takes optional arguments to
  - Include / Exclude field and/or property names as a comma-separated list or array

```
@Canonical
class Person {
  String firstName
  String lastName
  String email
}
```

Some useful AST transformations

@Immutable

- Class is made final
- Creates constructors and getters
- All fields are private
- Dates, Cloneables, and arrays are defensively copied on the way in and out
- Immutable types, like primitives and wrappers are allowed
- Fields that are enums or @Immutable are allowed
- Properties must themselves be immutable
- See GroovyDocs for details

Some useful AST transformations

Quiz:

1. What is the visibility of a class with no visibility modifier?
2. What is the visibility of attributes with no visibility modifier?
3. What is the visibility of methods with no visibility modifier?
4. How do we make something package protected?

Some useful AST transformations

@PackageScope

- On a class, makes class package protected
- On a field, makes field package protected
- On a method, makes field package protected
- See GroovyDocs for details

Some useful AST transformations

@TypeChecked

- On a class or method, causes Groovy compiler to use compile time checks in the style of Java

@CompileStatic

- Can be used on type, c'tor, method, field, local variable or even package declaration
- Same as @TypeChecked, except also does static compilation bypassing Groovy Meta-Object Protocol (MOP)
- You lose all dynamic behaviors with this, but get performance comparable to Java since dynamic method dispatch is bypassed
- Especially useful in upcoming Groovy 2.4.0 support of Android development

See GroovyDocs for details

Understanding the Groovy Meta-Object Protocol (MOP)

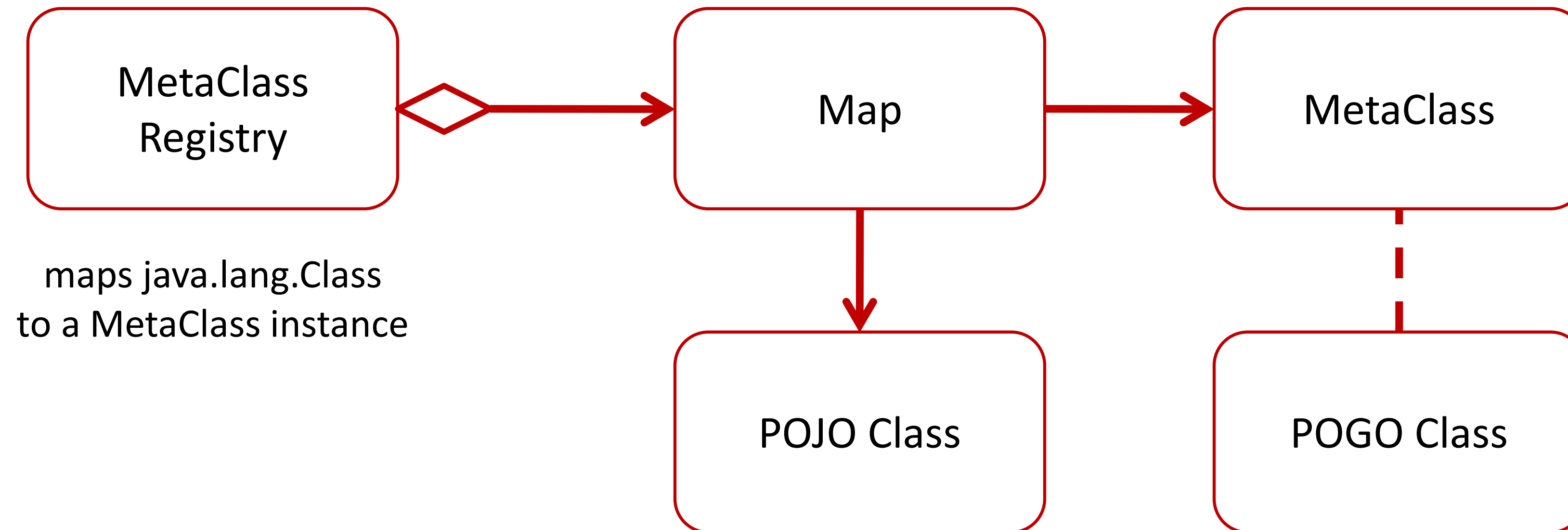
Groovy is a dynamic language

This means:

- Methods and object references are resolved at runtime
- The compiler gives us fewer errors since “missing” things may actually be valid at runtime
- Consider this:

```
class Person {
  String firstName
  String lastName
}
// somewhere else in the code…
new Person(firstName: ‘Fred’, lastName: ‘Flintstone’).save()
// MethodMissingException thrown if save() isn’t available at runtime
```

How does Groovy MOP Work?

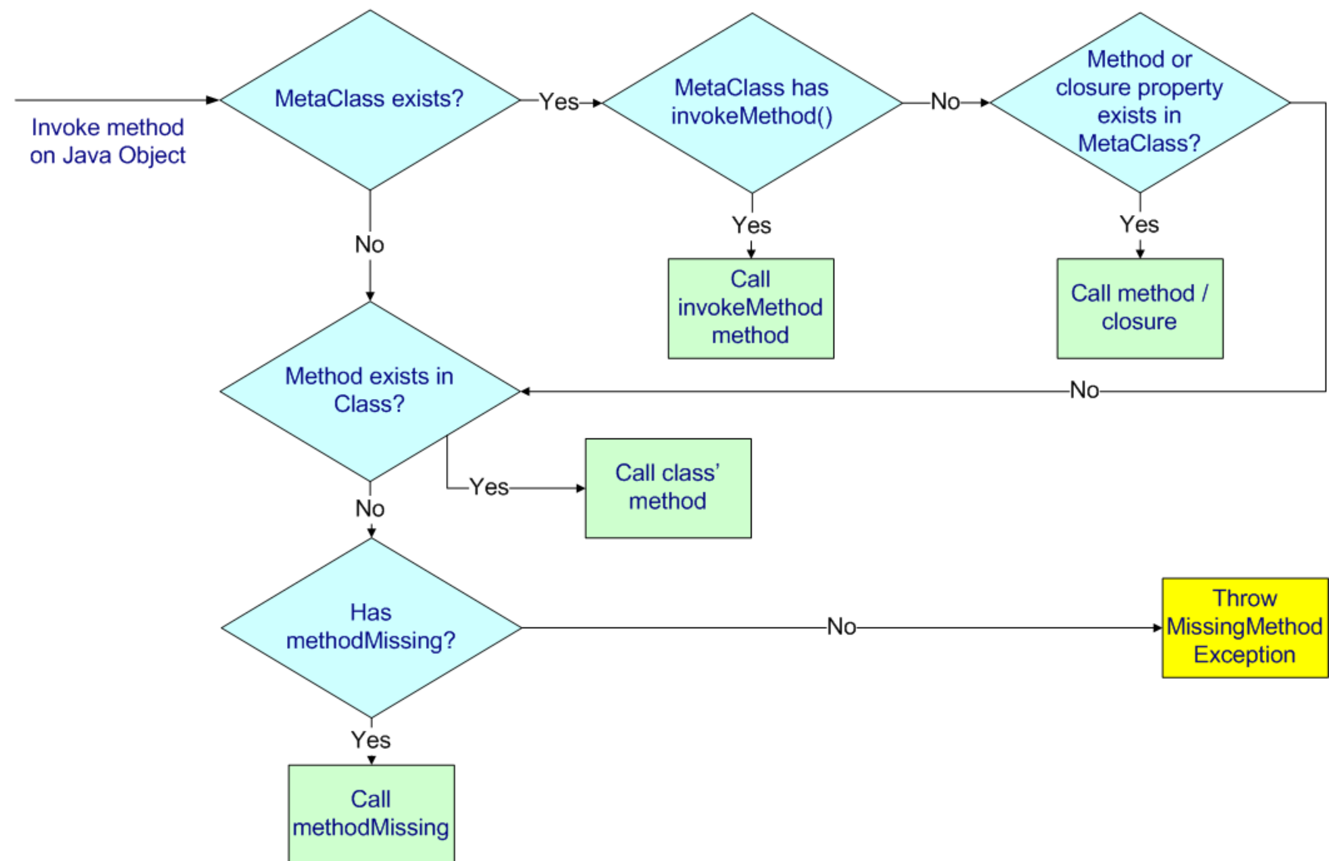


- To access the MetaClass for a Java object, Groovy queries the MetaClassRegistry via getMetaClass(Class) method
  - You can, too:

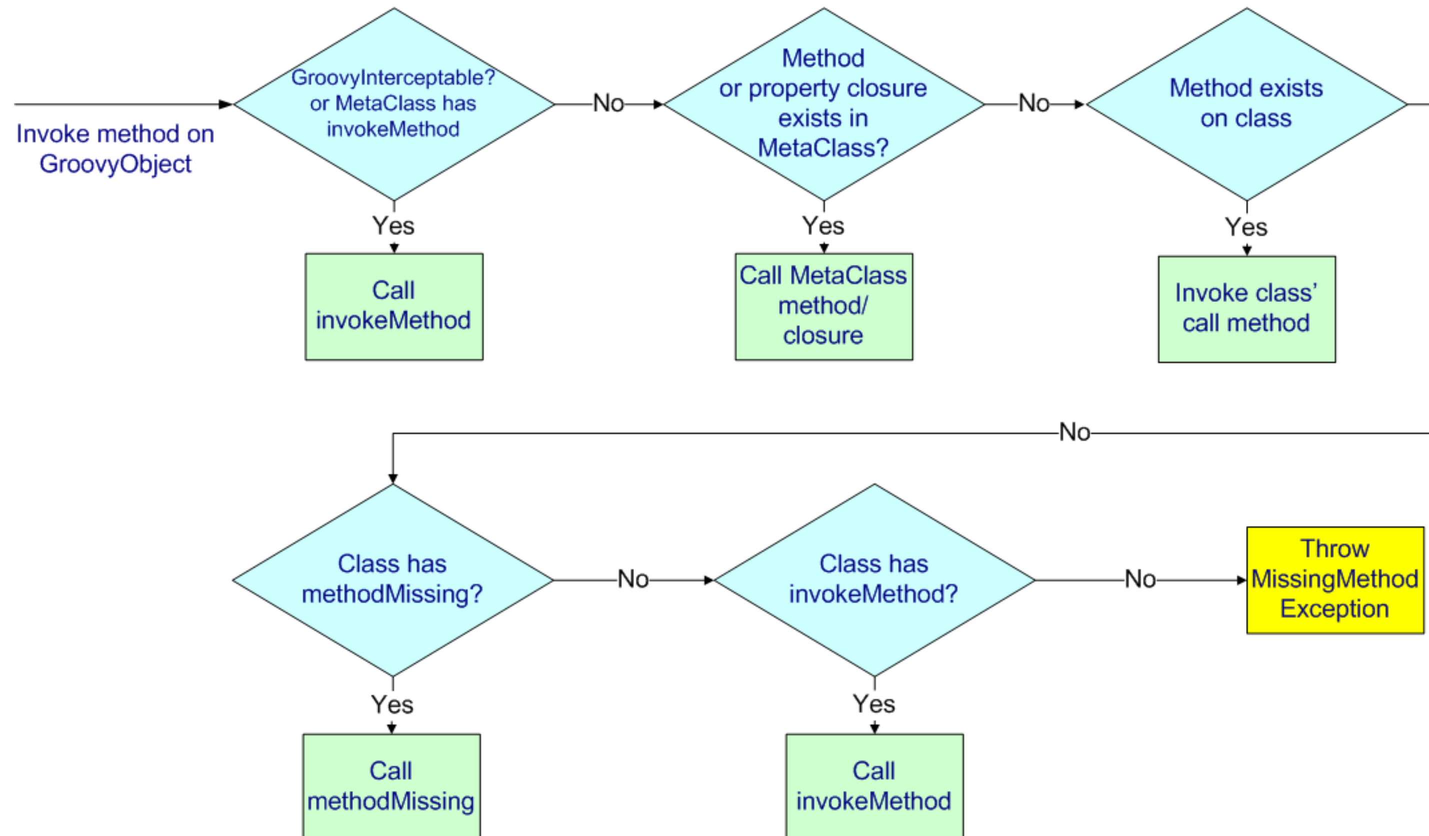
```
GroovySystem.metaClassRegistry.metaClass(java.lang.Integer)
```

- Groovy objects have direct access to their MetaClass object

Java method call under Groovy RT



Groovy method call under Groovy RT



Responsive Synthesis

With the MetaClass, we can add behavior to existing classes or instances with that behavior being available at runtime

The ultimate meta-programming is creating the code to add new behaviors at runtime

- Behaviors we don't even know about at compile time
- Behaviors that come into being based on runtime stimulus

Grails GORM finders is a first rate example

- `Person.findAllByLastNameLike('Flint%')`

Builders and Slurpers

Because of Groovy's dynamic nature, combined with its special syntax for closures, we can create elegant DSLs using (almost) plain language constructs to "Build" markup

Let's take a look at two:

- XMLMarkupBuilder
- JsonBuilder

Builders and Slurpers

It would be no fun if we could easily build XML or Json using a MarkupBuilder, but not as easily read it in

For that, let's use the XMLSlurper and JsonSlurper …

Part IV Self-Examination

1. AST in “AST Transformation” stands for ___ ___ ___
2. AST Transformations (ASTx) happen at compile time. (T/F)
3. Describe what affect the @Canonical ASTx has on a class
4. Describe what affect the @Immutable ASTx has on a class
5. What’s the upside to using @CompileStatic on a class?
6. What’s the downside to using @CompileStatic on a class?
7. If we add a Closure named toString to the metaClass of the Integer class (i.e `Integer.metaClass.toString = {…}`), will the Integer class’ toString or our closure be called when we execute 5.toString()?
8. If a Groovy class implements _______ then all method calls will be dispatched to the invokeMethod method
9. To easily create JSON from some data, use the _________
10. To easily create XML from some data, use the _________
11. To access elements of JSON in an object-like way, use the ______
12. To access elements of XML in an object-like way, use the ______

Part IV Labs

Please complete the Lab4_1xx labs

Time permitting, you may go on to the Lab4_2xx, 3xx, and 4xx labs.

However, please complete the easier labs before turning to the more challenging ones as the later labs build on concepts practiced in the earlier labs.

Fini

Let's connect on LinkedIn:

http://www.linkedin.com/in/jackfrosch

Appendix A
Useful Links

- http://groovy.codehaus.org
- http://radio-weblogs.com/0112098/2003/08/29.html
- http://glaforge.appspot.com/article/groovy-s-birthday-and-news
- http://groovy.codehaus.org/Differences+from+Java
- http://groovy.codehaus.org/Groovy+Truth
- http://groovy.codehaus.org/Operators
- http://groovy.codehaus.org/Building+AST+Guide
- http://groovy.codehaus.org/gapi/groovy/transform/package-summary.html
- http://groovy.codehaus.org/Builders
- http://groovy.codehaus.org/gapi/groovy/json/JsonBuilder.html
- http://groovy.codehaus.org/Building+AST+Guide
- http://groovy.codehaus.org/gapi/groovy/transform/package-summary.html
- http://groovy.codehaus.org/Builders
- http://groovy.codehaus.org/gapi/groovy/json/JsonBuilder.html

Appendix B
Self-Examination Answers

Part I

1. In what year was Groovy announced? ans. **2003**
2. What GA release of Groovy first introduced Traits? ans. **v2.3**
3. Groovy is a strongly typed statically compiled language. ans. **False!**
4. Groovy objects are java.lang.Objects. ans. **True**
5. All Groovy objects implement what interface? ans. **groovy.lang.GroovyObject**
6. All code you write in Groovy must be inside a class. ans. **False**
7. The ‘A’ literal specified with single quotes is what type? ans. **java.lang.String**
8. The “A-$code” literal is an instanceof groovy.lang.GString. ans. **True**
9. A triple quoted literal is called a h _ _ _d _ _ ans. **heredoc**
10. A method call to a non-existent method results in a compiler error. ans. **False!**
11. Given a1 == a2 compare whether a1 and a2 object references are the same. ans. **False - a.equals(b)**

Appendix B
Self-Examination Answers

Part II

1. Java overrides the + operator for Strings, but not BigDecimals. Adding BigDecimals a and b requires us to use a.add(b) instead of a+b. This is an example of _____ complexity imposed by the language. ans. **accidental**
2. The following are equivalent: println('Hello') println 'Hello' ans. **True**
3. If a method takes no arguments, parentheses are optional when calling the method. ans. **False**
4. When returning a value from a method the return statement is always required ans. **False – return is usually not required**
5. In Groovy truth, a non-null, but empty list is evaluated as true. ans. **False**
6. In Groovy truth, a non-zero Integer is evaluated as true. ans. **False**
7. Using the ____ operator Groovy operator we can shorten this common Java idiom: `String x = name != null ? name : "Anon"` **ans. Elvis (`String x = name ?: "Anon"`)**
8. Closures are behaviors that are first class types that can be passed around in the system. **ans. True**
9. The following declares a ___: def x = [1,3,5,7] ans. **List**
10. The following declares a _____ : def y = 1..99 ans. **Range**
11. The following declares a _____ : def z = [a:2, b:4, c: 6] ans. **Map**

Appendix B
Self-Examination Answers

Part III

1. To overload the + operator on a class, simply implement a method named ____ ans. **plus**
2. What is the result of this: `['Santa','Claus','Knows']*.size()` **ans. [5,5,5]**
3. What is the result of this: `['A','B','C'].join(',')` **ans. "A,B,C"**
4. What is the result of this: `2**3` **ans. 8**
5. What is the result of this: `' banner '.center(5, '*')` **ans. "***** banner *****"**
6. As of Groovy v2.3, ***trait*** is a new keyword ans. **True**
7. Do classes extend or implement a trait? ans. **extend: class A extends TraitB**
8. Like interfaces, traits can extend multiple traits. ans. **False**
9. Like a class, traits can implement interfaces ans. **True**
10. Traits can implement other traits ans. **True**

Appendix B
Self-Examination Answers

Part IV

1. AST in "AST Transformation" stands for ___ ___ ___ ans. **Abstract Syntax Tree**
2. AST Transformations (ASTx) happen at compile time. (T/F) ans. **True**
3. Describe what affect the @Canonical ASTx has on a class ans. **Equivalent to @TupleConstructor, @EqualsAndHashCode, @ToStri**ng
4. Describe what affect the @Immutable ASTx has on a class. ans. Class is made final; Creates constructors and getters; All fields are private
5. What's the upside to using @CompileStatic on a class? ans. **Better performance**
6. What's the downside to using @CompileStatic on a class? ans. **Unable to use dynamic behaviors**

Appendix B
Self-Examination Answers

Part IV (cont'd)

7. If we add a Closure named toString to the metaClass of the Integer class (i.e **Integer.metaClass.toString = {…}**), will the Integer class' toString or our closure be called when we execute 5.toString()? ans. **The metaClass closure will be called**

8. If a Groovy class implements ________ then all method calls will be dispatched to the invokeMethod method. ans. **GroovyInterceptable**

9. To easily create JSON from some data, use the ___________ ans. **JsonBuilder**

10. To easily create XML from some data, use the ___________ ans. **XMLMarkupBuilder**

11. To access elements of JSON in an object-like way, use the _______ ans. **JsonSlurper**

12. To access elements of XML in an object-like way, use the _______ ans. **XMLSlurper**