Introduction to Groovy

Part IV – More Groovy Basics

GROOVY… WHERE LESS REALLY IS MORE

Quick Review of Part III

Groovy Operator Overloading

More java.util Collections JDK enhancements

java.lang.String JDK enhancements

java.lang.Integer JDK enhancements

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

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

@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

Some useful AST Transformations

There are several more AST Transformations in the Groovy SDK

- See http://groovy.codehaus.org/gapi/groovy/transform/package-summary.html

The Groovy community also produces some open source ones

Understanding the Groovy Meta-Object Protocol (MOP)

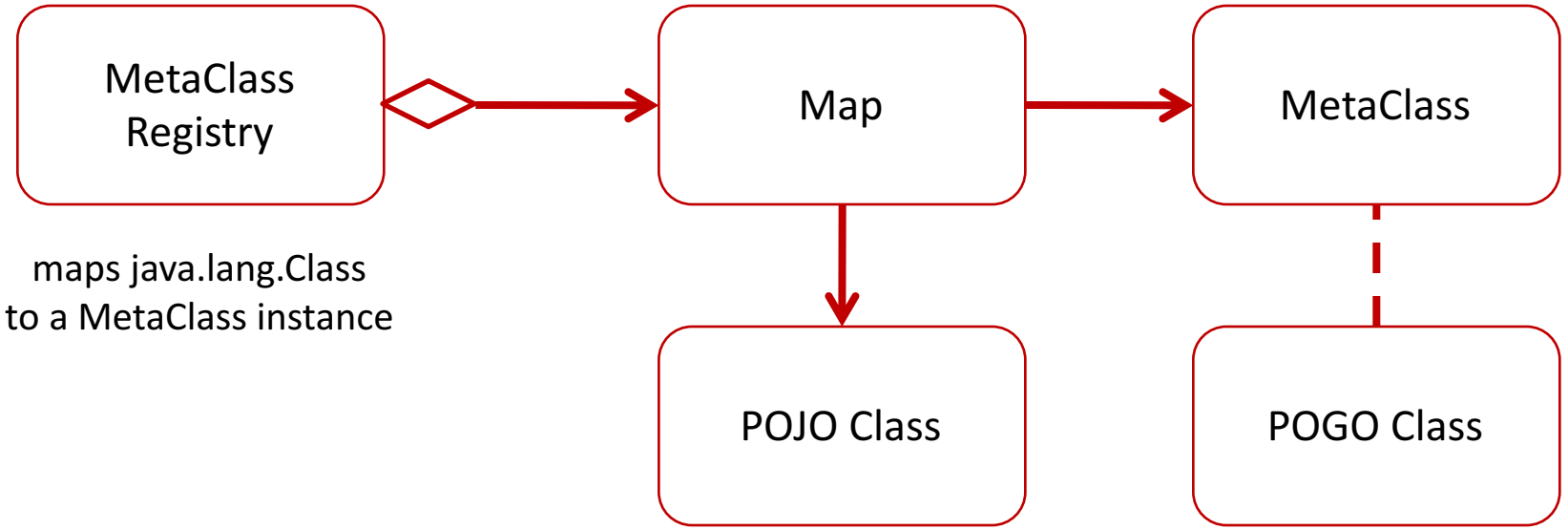
Groovy is a dynamic language

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?

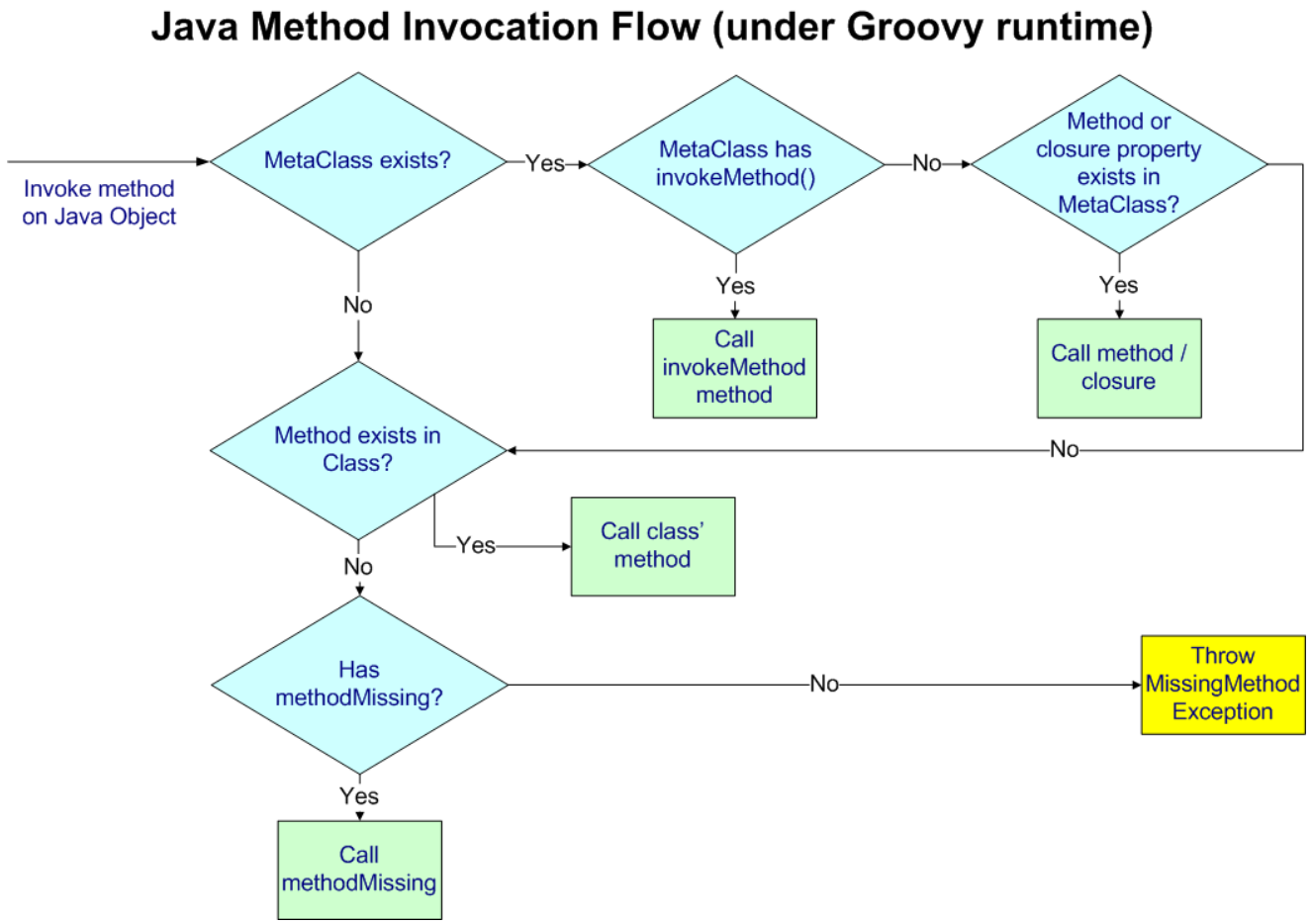


- To access the MetaClass for a Java object, Groovy queries the MetaClassRegistry via getMetaClass(Class) method
  - You can, too:

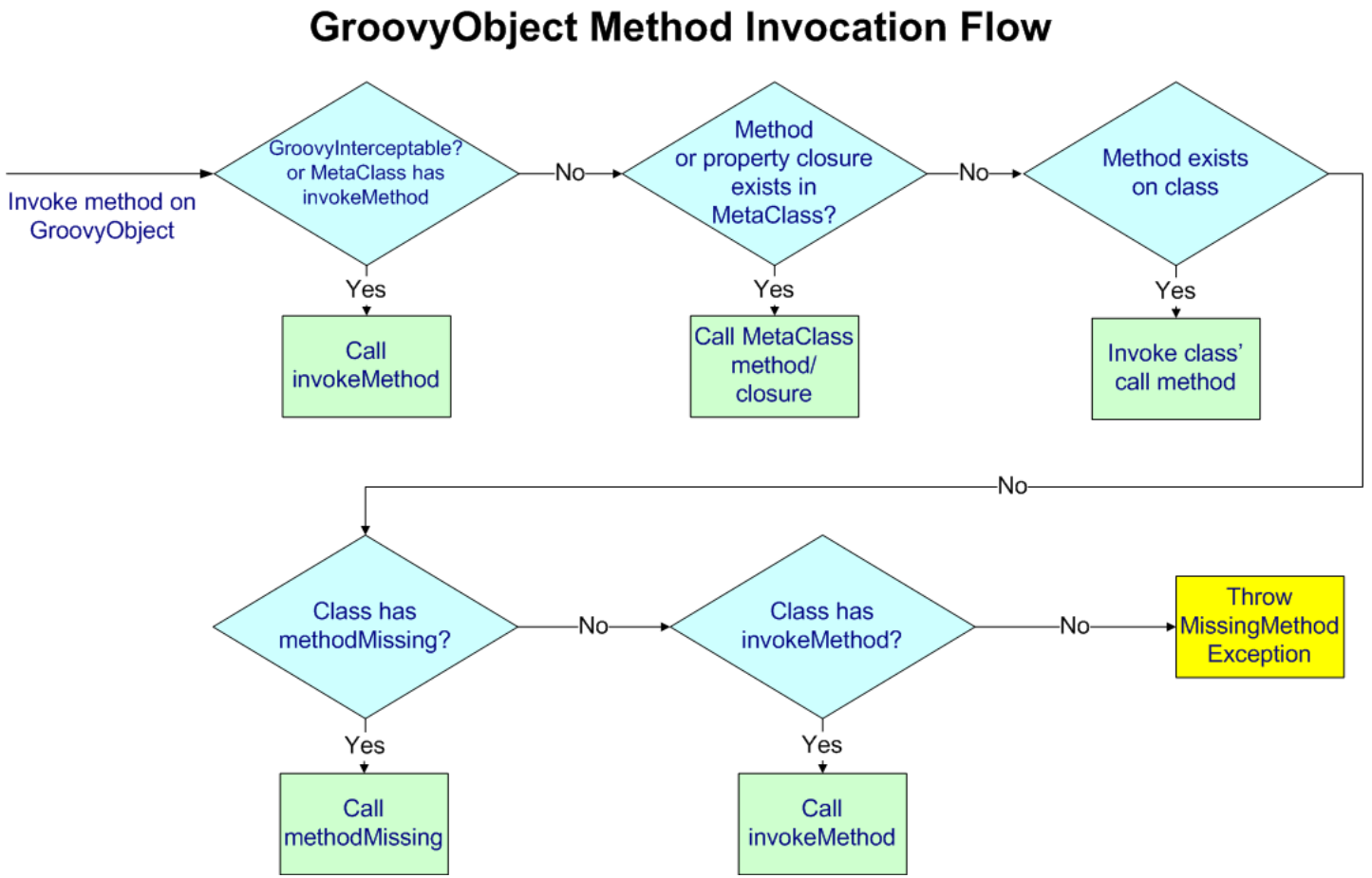
```
GroovySystem.metaClassRegistry.metaClass(java.lang.Integer)
```

- 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 …

Adding Groovy to Maven

Groovy comes with a groovy-all JAR

Adding the dependency to Maven POM looks like this:

```
<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy-all</artifactId>
  <version>${groovyVersion}</version>
  <!-- Uncomment line below to use invokedynamic ver of Groovy
       (requires Java 7 or higher). -->
  <!--<classifier>indy</classifier>-->
</dependency>
```

Adding Groovy to Maven

Groovy code is compiled with a Groovy aware compiler

- groovyc is the native *joint* compiler shipped with Groovy
- There's also a Groovy-Eclipse compiler plugin for Maven
- IDEs may use their own compiler or have plugins wrapping one of the standard solutions
- This page has a nice discussion of comparing different build solutions: http://docs.codehaus.org/display/GMAVENPLUS/Choosing+Your+Build+Tool

Adding Groovy to Maven

Using Groovy-Eclipse compiler plugin:

http://groovy.codehaus.org/Compiling+With+Maven2

Adding Groovy to Gradle

```
// build.gradle
apply plugin: 'eclipse‘ // here’s a better ‘idea’
apply plugin: 'groovy‘
repositories { mavenCentral() }
dependencies {
  compile 'org.codehaus.groovy:groovy-all:2.3.6‘
  testCompile 'junit:junit:4.11‘
}
```

Using Groovy with Your Favorite IDEs

Eclipse supports Groovy with Groovy Plugin

- http://groovy.codehaus.org/Eclipse+Plugin

If you're an Eclipse user, I highly recommend the Groovy Grails Tool Suite from Spring

- http://spring.io/tools/ggts

IDEA supports Groovy out of the box

- http://www.jetbrains.com/idea/features/groovy.html

NetBeans supports Groovy with a plugin

- http://groovy.codehaus.org/NetBeans+Plugin

What's Next

At our next meeting October 18th, we will have a talk on Grails

- I'll need someone to set up WebEx as I'll be dialing in

Resources & Links

Groovy

- http://groovy.codehaus.org
- 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