Groovy AST Transformations

Dr Paul King
@paulk_asert

http://slideshare.net/paulk_asert/groovy-transforms
https://github.com/paulk-asert/groovy-transforms

Topics

- **Introduction**
- Built-in AST Transforms
- Writing your own transforms

Parsing Summary

> groovy MyScript.groovy
> groovyc MyScript.groovy
> groovysh
> groovyConsole

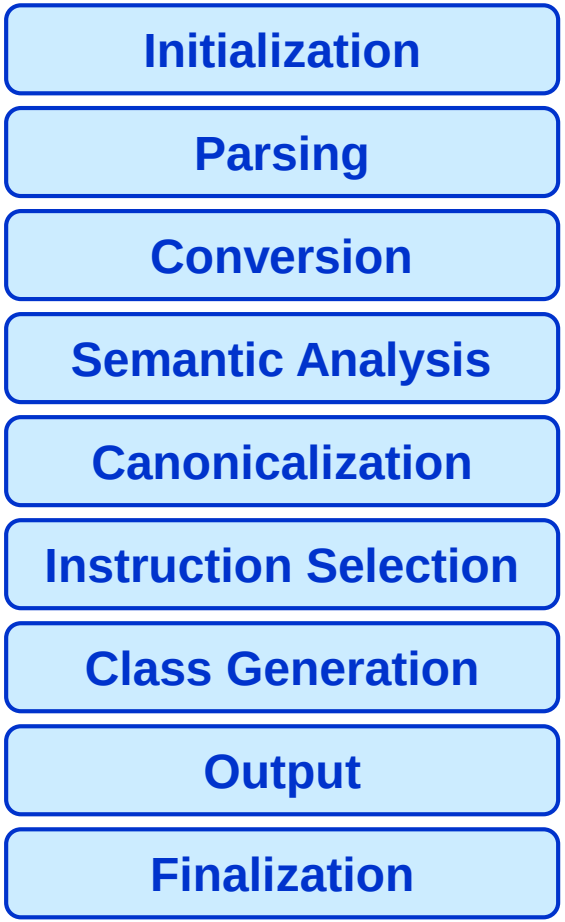
MyScript.groovy

```
println "Howdy Y'all"
```

```
BlockStatement
-> ReturnStatement
  -> MethodCallExpression
    -> VariableExpression("this")
    -> ConstantExpression("println")
    -> ArgumentListExpression
      -> ConstantExpression("Howdy Y'all")
```

```
public run()
  ...
  L1
   ALOAD 1
   LDC 1
   AALOAD
   ALOAD 0
   LDC "Howdy Y'all"
   INVOKEINTERFACE callCurrent()
   ARETURN
  ...
```

Parsing Summary



- 9 phase compiler
  - Early stages: *read source code and convert into a sparse syntax tree*
  - Middle stages: *iteratively build up a more dense and information rich version of the syntax tree*
  - Later stages: *check the tree and convert it into byte code/class files*

Parsing Summary

- Initialization
- Parsing
- Conversion
- Semantic Analysis
- Canonicalization
- Instruction Selection
- Class Generation
- Output
- Finalization

```
@ToString
class Greeter {
  String message = "Howdy Y'all"
  void greet() {
    println message
  }
}
```

ClassNode: Greeter

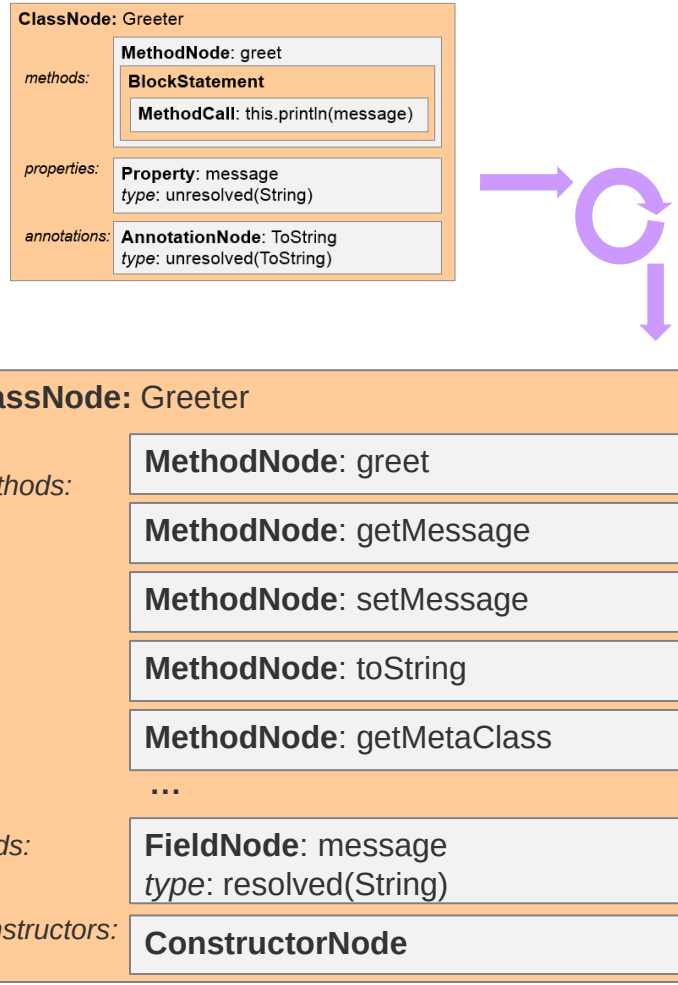
methods:
- **MethodNode**: greet
  - **BlockStatement**
    - **MethodCall**: this.println(message)

properties:
- **Property**: message
  type: unresolved(String)

annotations:
- **AnnotationNode**: ToString
  type: unresolved(ToString)

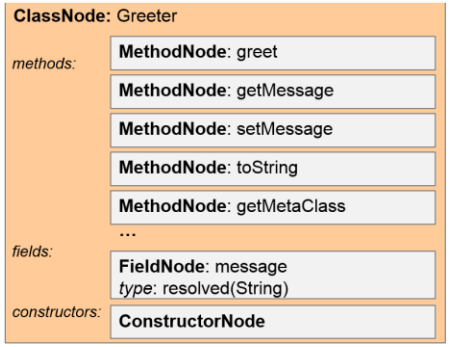
Parsing Summary

- Initialization
- Parsing
- Conversion
- **Semantic Analysis**
- **Canonicalization**
- **Instruction Selection**
- Class Generation
- Output
- Finalization



Parsing Summary

- Initialization
- Parsing
- Conversion
- Semantic Analysis
- Canonicalization
- Instruction Selection
- **Class Generation**
- **Output**
- **Finalization**




```
public greet()V
  ...
  L1
    ...
    ALOAD 0
    GETFIELD Greeter.message
    INVOKEINTERFACE callCurrent()
    POP
    ...
```

Parsing Summary

- Initialization
- Parsing
- Conversion
- Semantic Analysis
- Canonicalization
- Instruction Selection
- Class Generation
- Output
- Finalization

Global Transformations

Local Transformations

Topics

- Introduction
- ➢ **Built-in AST Transforms**
- Writing your own transforms

@ToString


```groovy
class Detective {
    String firstName, lastName
    String toString() {
        def _result = new StringBuilder()
        _result.append('Detective(')
        _result.append(this.firstName)
        _result.append(', ')
        _result.append(this.lastName)
        _result.append(')')
        return _result.toString()
    }
}
```

```groovy
@groovy.transform.ToString
class Detective {
    String firstName, lastName
}

def d = new Detective(firstName: 'Sherlock', lastName: 'Holmes')
assert d.toString() == 'Detective(Sherlock, Holmes)'
```

@ToString annotation parameters…

Parameter Name	Purpose
excludes	Exclude certain properties from `toString()` by specifying the property names as a comma separated list or literal list of String name values. By default, all properties are included. Incompatible with “includes”.
includes	Include just the specified properties by specifying the property names as a comma separated list or literal list of String name values. Incompatible with “excludes”.
includeSuper	Include the `toString()` for the super class by setting to true. Default: false.
includeNames	Include the names of the properties by setting this parameter to true. Default: false.

…@ToString annotation parameters

Parameter Name	Purpose
includeFields	Include the class's fields, not just the properties, by setting this parameter to true. Default: false.
ignoreNulls	Exclude any properties that are null. By default null values will be included.
includePackage	Set to false to print just the simple name of the class without the package. By default the package name is included.
cache	Set to true to cache `toString()` calculations. Use only for immutable objects. By default the `toString()` is recalculated whenever the `toString()` method is called.

@ToString (using parameters)

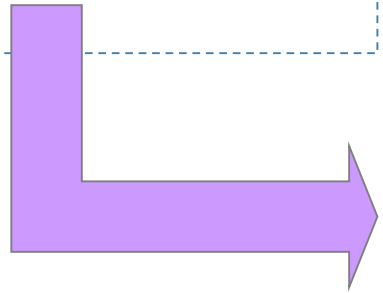

```
@ToString(ignoreNulls=true, excludes='lastName', includeNames=true,
        includePackage=false, includeFields=true)
class Detective {
    String firstName, lastName
    List clues
    private nemesis = 'Moriarty'
}

def d = new Detective(firstName: 'Sherlock', lastName: 'Holmes')
assert d.toString() ==
        'Detective(firstName:Sherlock, nemesis:Moriarty)'
```

@EqualsAndHashCode


```
@EqualsAndHashCode
class Actor {
    String firstName, lastName
}

def a1 = new Actor(firstName: 'Ian', lastName: 'McKellen')
def a2 = new Actor(firstName: 'Ian', lastName: 'McKellen')
assert !(a1.is(a2))
assert a1 == a2
```



EqualsAndHashCodeASTTransformation

@EqualsAndHashCode


```
class Actor {
    String firstName, lastName
    int hashCode() {
        def _result = HashCodeHelper.initHash()
        _result = HashCodeHelper.updateHash(_result, this.firstName)
        _result = HashCodeHelper.updateHash(_result, this.lastName)
        return _result
    }

    boolean canEqual(other) {
        return other instanceof Actor
    }
    ...
```

@EqualsAndHashCode


```
    ...
    boolean equals(other) {
        if (other == null) return false
        if (this.is(other)) return true
        if (!( other instanceof Actor)) return false
        Actor otherTyped = (Actor) other
        if (!(otherTyped.canEqual(this))) return false
        if (!(this.getFirstName().is(otherTyped.getFirstName()))) {
            if (!(this.getFirstName() == otherTyped.getFirstName())) {
                return false
            }
        }
        if (!(this.getLastName().is(otherTyped.getLastName()))) {
            if (!(this.getLastName() == otherTyped.getLastName())) {
                return false
            }
        }
        return true
    }
}
```

@EqualsAndHashCode annotation parameters…

Parameter Name	Purpose
excludes	Exclude certain properties from the calculation by specifying them as a comma separated list or literal list of String name values. This is commonly used with an object that has an 'id' field. By default, no properties are excluded. Incompatible with “includes”
includes	Include only a specified list of properties by specifying them as a comma separated list or literal list of String name values. Incompatible with “excludes”.
cache	Set to true to cache `hashCode()` calculations. Use only for immutable objects. By default the `hashCode()` is recalculated whenever the `hashCode()` method is called.

…@EqualsAndHashCode annotation parameters

Parameter Name	Purpose
callSuper	Include properties from the super class by setting this parameter to true. By default, the super class is not used as part of the calculation.
includeFields	Include the class's fields, not just the properties, in the calculation by setting this parameter to true. By default, fields are not taken into account.
useCanEqual	Set to false to disable generation of a `canEqual()` method to be used by `equals()`. By default the `canEqual()` method is generated. The *canEqual* idiom provides a mechanism for permitting equality in the presence of inheritance hierarchies. For immutable classes with no explicit super class, this idiom is not required.

@TupleConstructor

- A traditional positional arguments constructor

```
@TupleConstructor
class Athlete {
    String firstName, lastName
}

def a1 = new Athlete('Michael', 'Jordan')
def a2 = new Athlete('Michael')
def a3 = new Athlete(firstName: 'Michael')

assert a1.firstName == a2.firstName
assert a2.firstName == a3.firstName
```

@Canonical

- Combines @ToString, @EqualsAndHashCode and @TupleConstructor

```
@Canonical
class Inventor {
    String firstName, lastName
}

def i1 = new Inventor('Thomas', 'Edison')
def i2 = new Inventor('Thomas')
assert i1 != i2
assert i1.firstName == i2.firstName
assert i1.toString() == 'Inventor(Thomas, Edison)'
```

@Lazy…

- Safe and efficient deferred construction

```
// nominally expensive resource with stats
class Resource {
    private static alive = 0
    private static used = 0
    Resource() { alive++ }
    def use() { used++ }
    static stats() { "$alive alive, $used used" }
}
```

- Understands double checked-locking and holder class idioms

...@Lazy

```
class ResourceMain {
    def res1 = new Resource()
    @Lazy res2 = new Resource()
    @Lazy static res3 = { new Resource() }()
    @Lazy(soft=true) volatile Resource res4
}

new ResourceMain().with {
    assert Resource.stats() == '1 alive, 0 used'
    res2.use()
    res3.use()
    res4.use()
    assert Resource.stats() == '4 alive, 3 used'
    assert res4 instanceof Resource
    def expected = 'res4=java.lang.ref.SoftReference'
    assert it.dump().contains(expected)
}
```

@InheritConstructors

- Reduced boiler-plate for scenarios where parent classes have multiple constructors (e.g. Exceptions & PrintWriter)

```
@InheritConstructors
class MyPrintWriter extends PrintWriter { }

def pw1 = new MyPrintWriter(new File('out1.txt'))
def pw2 = new MyPrintWriter('out2.txt', 'US-ASCII')
[pw1, pw2].each {
    it << 'foo'
    it.close()
}
assert new File('out1.txt').text == new
File('out2.txt').text
['out1.txt', 'out2.txt'].each{ new File(it).delete() }
```

@Sortable…

- Reduced boiler-plate for Comparable classes including specialised Comparators

```
@Sortable(includes = 'last,initial')
class Politician {
    String first
    Character initial
    String last

    String initials() { first[0] + initial + last[0] }
}
```

…@Sortable


```
// @Sortable(includes = 'last,initial') class Politician { ... }

def politicians = [
    new Politician(first: 'Margaret', initial: 'H', last: 'Thatcher'),
    new Politician(first: 'George', initial: 'W', last: 'Bush')
]

politicians.with {
    assert sort()*.initials() == ['GWB', 'MHT']
    def comparator = Politician.comparatorByInitial()
    assert toSorted(comparator)*.initials() == ['MHT', 'GWB']
}
```

Trip Highlights

Groovy and "fluent-api" builders

- A common idiom in recent times for Java is to use an inner helper class and accompanying "fluent-api" to reduce ceremony when creating Java classes with many parameters
  - But Groovy's named parameters greatly reduces this need

```
class Chemist {
    String first
    String last
    int born
}

def c = new Chemist(first: "Marie", last: "Curie", born: 1867)
assert c.first == "Marie"
assert c.last == "Curie"
assert c.born == 1867
```

@Builder…

- But if you need Java integration or want improved IDE support…

```
import groovy.transform.builder.Builder

@Builder
class Chemist {
    String first
    String last
    int born
}

def builder = Chemist.builder()
def c = builder.first("Marie").last("Curie").born(1867).build()
assert c.first == "Marie"
assert c.last == "Curie"
assert c.born == 1867
```

…@Builder…

- And it supports customizable building strategies

Strategy	Description
DefaultStrategy	Creates a nested helper class for instance creation. Each method in the helper class returns the helper until finally a `build()` method is called which returns a created instance.
SimpleStrategy	Creates chainable setters, i.e. each setter returns the object itself after updating the appropriate property.
ExternalStrategy	Allows you to annotate an explicit builder class while leaving some buildee class being built untouched. This is appropriate when you want to create a builder for a class you don't have control over, e.g. from a library or another team in your organization.
InitializerStrategy	Creates a nested helper class for instance creation which when used with `@CompileStatic` allows type-safe object creation.

…@Builder

- Type-safe construction using phantom types (*if* you need it)

```
@Builder(builderStrategy=InitializerStrategy)
@Immutable
class Chemist {
   String first, last
   int born
}

@CompileStatic
def solution() {
   def init = Chemist.createInitializer().first("Marie").last("Curie").born(1867)
   new Chemist(init).with {
      assert first == "Marie"
      assert last == "Curie"
      assert born == 1867
   }
}

solution()
```

@Delegate (motivation)

- Anything wrong with this?

```
class NoisySet extends HashSet {
    @Override
    boolean add(item) {
        println "adding $item"
        super.add(item)
    }

    @Override
    boolean addAll(Collection items) {
        items.each { println "adding $it" }
        super.addAll(items)
    }
}
```

@Delegate (motivation)

- Anything wrong with this?
- Could we fix this implementation?

```
class NoisySet extends HashSet {
    @Override
    boolean add(item) {
        println "adding $item"
        super.add(item)
    }

    @Override
    boolean addAll(Collection items) {
        items.each { println "adding $it" }
        super.addAll(items)
    }
}
```

@Delegate (motivation)

- Anything wrong with this?
- Could we fix this implementation?
- What about using the delegate pattern written by hand?

```
class NoisySet extends HashSet {
    @Override
    boolean add(item) {
        println "adding $item"
        super.add(item)
    }

    @Override
    boolean addAll(Collection items) {
        items.each { println "adding $it" }
        super.addAll(items)
    }
}
```

@Delegate

- For declarative but flexible use of the delegate pattern

```
class NoisySet {
    @Delegate
    Set delegate = new HashSet()

    @Override
    boolean add(item) {
        println "adding $item"
        delegate.add(item)
    }

    @Override
    boolean addAll(Collection items) {
        items.each { println "adding $it" }
        delegate.addAll(items)
    }
}
```

```
Set ns = new NoisySet()
ns.add(1)
ns.addAll([2, 3])
assert ns.size() == 3
```

@Delegate annotation parameters…

Parameter Name	Purpose
interfaces	Set this parameter to true to make the owner class implement the same interfaces as the delegate, which is the default behavior. To make the owner *not* implement the delegate interfaces, set this parameter to false.
deprecated	Set this parameter to true to have the owner class delegate methods marked as `@Deprecated` in the delegate. By default `@Deprecated` methods are not delegated.
methodAnnotations	Set to true if you want to carry over annotations from the methods of the delegate to your delegating method. By default, annotations are not carried over. Currently Closure annotation members are not supported.
parameterAnnotations	Set to true if you want to carry over annotations from the method parameters of the delegate to your delegating method. By default, annotations are not carried over. Currently Closure annotation members are not supported.

…@Delegate annotation parameters

Parameter Name	Purpose
excludes	List of method and/or property names to exclude when delegating.
excludeTypes	List of interfaces containing method signatures to exclude when delegating.
includes	List of method and/or property names to include when delegating.
includeTypes	List of interfaces containing method signatures to exclude when delegating.

- Only one of 'includes', 'includeTypes', 'excludes' or 'excludeTypes' should be used.

@Memoized

- For making pure functions more efficient

```
class Calc {
  def log = []

  @Memoized
  int sum(int a, int b) {
    log << "$a+$b"
    a + b
  }
}
```

```
new Calc().with {
  assert sum(3, 4) == 7
  assert sum(4, 4) == 8
  assert sum(3, 4) == 7
  assert log.join(' ') == '3+4 4+4'
}
```

@Memoized

- For making pure functions more efficient

```
class Calc {
  def log = []

  @Memoized
  int sum(int a, int b) {
    log << "$a+$b"
    a + b
  }
}
```

```
new Calc().with {
  assert sum(3, 4) == 7
  assert sum(4, 4) == 8
  assert sum(3, 4) == 7
  assert log.join(' ') == '3+4 4+4'
}
```

@TailRecursive…

- For unravelling recursion

```
import groovy.transform.TailRecursive

class RecursiveCalc {
    @TailRecursive
    int accumulate(int n, int sum = 0) {
        (n == 0) ? sum : accumulate(n - 1, sum + n)
    }
}

new RecursiveCalc().with {
    assert accumulate(10) == 55
}
```

…@TailRecursive

```
class RecursiveCalc {
    int accumulate(int n, int sum) {
        int _sum_ = sum
        int _n_ = n
        while (true) {
            if (_n_ == 0) {
                return _sum_
            } else {
                int __n__ = _n_
                int __sum__ = _sum_
                _n_ = __n__ - 1
                _sum_ = __sum__ + __n__
            }
        }
    }

    int accumulate(int n) { accumulate(n, 0) }
}
```

@Immutable

- For unchanging data structures

```
@Immutable
class Genius {
    String firstName, lastName
}

def g1 = new Genius(firstName: 'Albert', lastName: "Einstein")
assert g1.toString() == 'Genius(Albert, Einstein)'

def g2 = new Genius('Leonardo', "da Vinci")
assert g2.firstName == 'Leonardo'

shouldFail(ReadOnlyPropertyException) {
    g2.lastName = 'DiCaprio'
}
```

@Log @Log4j @Log4j2 @Commons @Slf4j

- For easy logging

```groovy
@groovy.util.logging.Log
class Database {
    def search() {
        log.fine(runLongDatabaseQuery())
    }

    def runLongDatabaseQuery() {
        println 'Calling database'
        /* ... */
        return 'query result'
    }
}

new Database().search()
```

@Synchronized

- For safe synchronization

```
class PhoneBook1 {
    private final phoneNumbers = [:]

    @Synchronized
    def getNumber(key) {
        phoneNumbers[key]
    }

    @Synchronized
    void addNumber(key, value) {
        phoneNumbers[key] = value
    }
}
```

@WithReadLock @WithWriteLock

- Declarative and efficient synchronization

```
class PhoneBook2 {
    private final phoneNumbers = [:]

    @WithReadLock
    def getNumber(key) {
        phoneNumbers[key]
    }

    @WithWriteLock
    def addNumber(key, value) {
        phoneNumbers[key] = value
    }
}
```

Trip Highlights

@AutoClone (Simple example)

- Easier cloning. With multiple styles supported: because one size doesn't fit all for cloning on the JVM

```
@AutoClone
class Chef1 {
    String name
    List<String> recipes
    Date born
}

def name = 'Heston Blumenthal'
def recipes = ['Snail porridge', 'Bacon & egg ice cream']
def born = Date.parse('yyyy-MM-dd', '1966-05-27')
def c1 = new Chef1(name: name, recipes: recipes, born: born)
def c2 = c1.clone()
assert c2.recipes == recipes
```

@AutoClone (Advanced example)


```
@TupleConstructor
@AutoClone(style=COPY_CONSTRUCTOR)
class Person {
    final String name
    final Date born
}

@TupleConstructor(includeSuperProperties=true, callSuper=true)
@AutoClone(style=COPY_CONSTRUCTOR)
class Chef2 extends Person {
    final List<String> recipes
}

def name = 'Jamie Oliver'
def recipes = ['Lentil Soup', 'Crispy Duck']
def born = Date.parse('yyyy-MM-dd', '1975-05-27')
def c1 = new Chef2(name, born, recipes)
def c2 = c1.clone()
assert c2.name == name
assert c2.born == born
assert c2.recipes == recipes
```

@AutoCloneStyle

Style	Description
CLONE	Adds a `clone()` method to your class. The `clone()` method will call `super.clone()` before calling `clone()` on each `Cloneable` property of the class. Doesn't provide deep cloning. Not suitable if you have final properties. (Default)
SIMPLE	Adds a `clone()` method to your class which calls the no-arg constructor then copies each property calling `clone()` for each `Cloneable` property. Handles inheritance hierarchies. Not suitable if you have final properties. Doesn't provide deep cloning.
COPY_CONSTRUCTOR	Adds a "copy" constructor, i.e. one which takes your class as its parameter, and a `clone()` method to your class. The copy constructor method copies each property calling `clone()` for each `Cloneable` property. The `clone()` method creates a new instance making use of the copy constructor. Suitable if you have final properties. Handles inheritance hierarchies. Doesn't provide deep cloning.
SERIALIZATION	Adds a `clone()` method to your class which uses serialization to copy your class. Suitable if your class already implements the `Serializable` or `Externalizable` interface. Automatically performs deep cloning. Not as time or memory efficient. Not suitable if you have final properties.

@AutoExternalize

```groovy
@AutoExternalize
@ToString
class Composer {
    String name
    int born
    boolean married
}

def c = new Composer(name: 'Wolfgang Amadeus Mozart',
        born: 1756, married: true)

def baos = new ByteArrayOutputStream()
baos.withObjectOutputStream{ os -> os.writeObject(c) }
def bais = new ByteArrayInputStream(baos.toByteArray())
def loader = getClass().classLoader
def result
bais.withObjectInputStream(loader) {
    result = it.readObject().toString()
}
assert result == 'Composer(Wolfgang Amadeus Mozart, 1756, true)'
```

@TimedInterrupt
@ThreadInterrupt
@ConditionalInterrupt

- For safer scripting
- Typically applied through compilation customizers to user scripts rather than directly used

@TimedInterrupt

```
@TimedInterrupt(value = 520L, unit = MILLISECONDS)
class BlastOff1 {
    def log = []
    def countdown(n) {
        sleep 100
        log << n
        if (n == 0) log << 'ignition'
        else countdown(n - 1)
    }
}
```

```
def b = new BlastOff1()
Thread.start {
    try {
        b.countdown(10)
    } catch (TimeoutException ignore) {
        b.log << 'aborted'
    }
}.join()
assert b.log.join(' ') == '10 9 8 7 6 aborted'
```

@ThreadInterrupt

```
@ThreadInterrupt
class BlastOff2 {
    def log = []
    def countdown(n) {
        Thread.sleep 100
        log << n
        if (n == 0) log << 'ignition'
        else countdown(n - 1)
    }
}
```

```
def b = new BlastOff2()
def t1 = Thread.start {
    try {
        b.countdown(10)
    } catch(InterruptedException ignore) {
        b.log << 'aborted'
    }
}
sleep 570
t1.interrupt()
t1.join()
assert b.log.join(' ') == '10 9 8 7 6 aborted'
```

@ConditionalInterrupt

```
@ConditionalInterrupt({ count <= 5 })
class BlastOff3 {
    def log = []
    def count = 10
    def countdown() {
        while (count != 0) {
            log << count
            count--
        }
        log << 'ignition'
    }
}
```

```
def b = new BlastOff3()
try {
    b.countdown()
} catch (InterruptedException ignore) {
    b.log << 'aborted'
}
assert b.log.join(' ') == '10 9 8 7 6 aborted'
```

Topics

- Introduction
- Built-in AST Transforms
- ➢ **Writing your own transforms**

Parsing Deep Dive

- **Initialization**
- Parsing
- Conversion
- Semantic Analysis
- Canonicalization
- Instruction Selection
- Class Generation
- Output
- Finalization

- Purpose

  Read source files/streams and configure compiler

- Key classes

  CompilerConfiguration
  CompilationUnit

Parsing Deep Dive

- Initialization
- **Parsing**
- Conversion
- Semantic Analysis
- Canonicalization
- Instruction Selection
- Class Generation
- Output
- Finalization

- Purpose

  Use (ANTLR) grammar to convert source code into token tree

- Key classes

  CompilationUnit GroovyLexer GroovyRecognizer GroovyTokenTypes

- CST Transforms
  - http://java.dzone.com/articles/groovy-antlr-plugins-better

Parsing Deep Dive

- Initialization
- Parsing
- **Conversion**
- Semantic Analysis
- Canonicalization
- Instruction Selection
- Class Generation
- Output
- Finalization

- Purpose

  Token tree converted into abstract syntax tree (AST) and is the first place where we can begin to write AST visitors
- Key classes

  AntlrParserPlugin EnumVisitor
- AST Transforms

  @Grab (global)

Parsing Deep Dive

- Initialization
- Parsing
- Conversion
- **Semantic Analysis**
- Canonicalization
- Instruction Selection
- Class Generation
- Output
- Finalization

- Purpose

  Resolves classes and performs consistency and validity checks beyond what the grammar can provide

- Key classes

  StaticVerifier ResolveVisitor StaticImportVisitor InnerClassVisitor, AnnotationCollector

- AST Transforms

  @Lazy @Builder @Field @Log @Memoized @PackageScope @TailRecursive @BaseScript

Parsing Deep Dive

- Initialization
- Parsing
- Conversion
- Semantic Analysis
- **Canonicalization**
- Instruction Selection
- Class Generation
- Output
- Finalization

- Purpose

  Finalizes the complete abstract syntax tree and typically the last point at which you want to run a transformation

- Key classes

  InnerClassCompletionVisitor
  EnumCompletionVisitor, TraitComposer

- AST Transforms

  @Bindable @Vetoable @Mixin @AutoClone
  @ConditionalInterrupt @ThreadInterrupt
  @TimedInterrupt @ListenerList @Canonical
  @Category @Delegate @Bindable
  @Vetoable @EqualsAndHashCode
  @AutoExternalize @Immutable
  @IndexedProperty @Synchronized
  @InheritConstructors @Sortable
  @WithReadLock @WithWriteLock
  @Singleton @Newify @ToString
  @TupleConstructor

Parsing Deep Dive

- Initialization
- Parsing
- Conversion
- Semantic Analysis
- Canonicalization
- **Instruction Selection**
- Class Generation
- Output
- Finalization

- Purpose

  Chooses an instruction set for the generated bytecode, e.g. Java 5 versus pre-Java 5

- AST Transforms

  @CompileStatic
  @TypeChecked

Parsing Deep Dive

- Initialization
- Parsing
- Conversion
- Semantic Analysis
- Canonicalization
- Instruction Selection
- **Class Generation**
- Output
- Finalization

- Purpose

  Creates bytecode based Class in memory

- Key classes

  OptimizerVisitor
  GenericsVisitor Verifier
  LabelVerifier
  ExtendedVerifier
  ClassCompletionVerifier
  AsmClassGenerator

Parsing Deep Dive

- Initialization
- Parsing
- Conversion
- Semantic Analysis
- Canonicalization
- Instruction Selection
- Class Generation
- **Output**
- Finalization

- Purpose

  Binary output (.class file) written to file system

Parsing Deep Dive

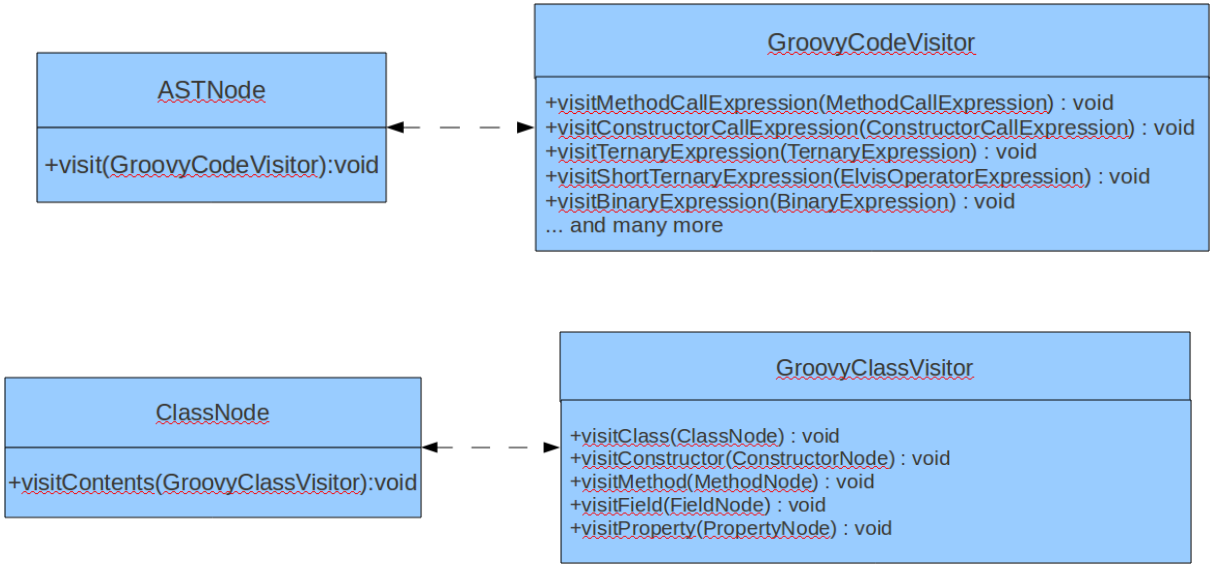
- Initialization
- Parsing
- Conversion
- Semantic Analysis
- Canonicalization
- Instruction Selection
- Class Generation
- Output
- **Finalization**

- Purpose

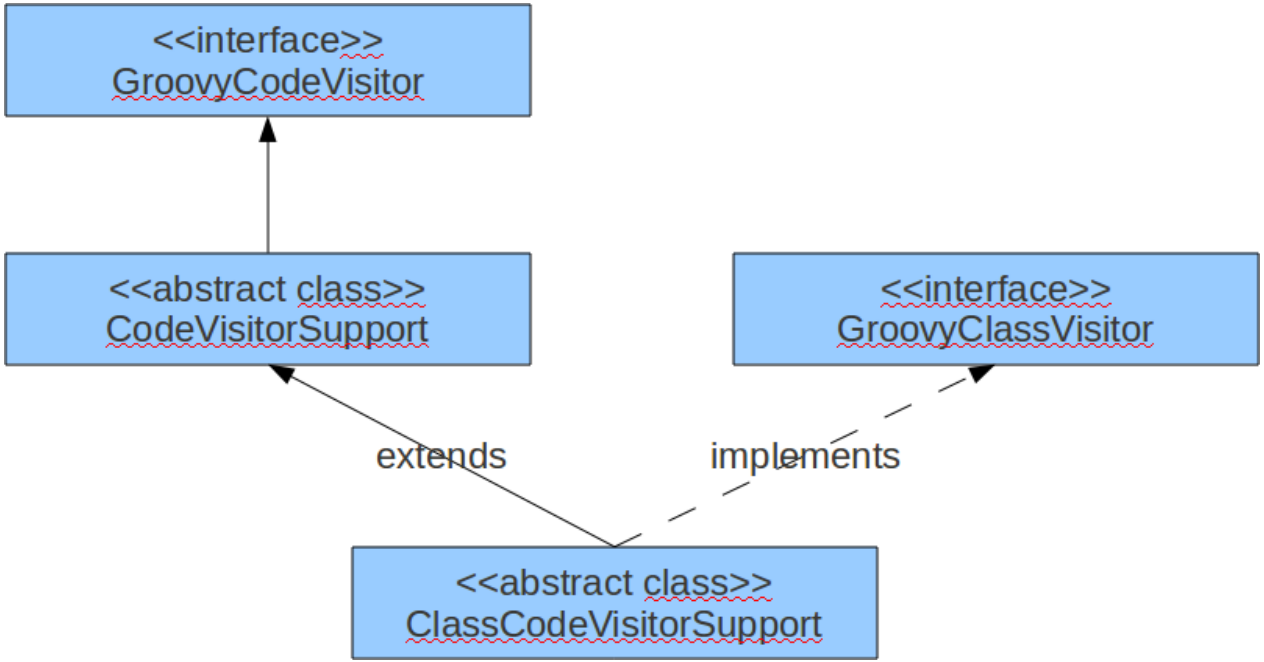
  Used to cleanup any resources no longer needed

Visitor Pattern…

- separates the object being walked (the tree) from the behavior of the walker (the visitor)



…Visitor Pattern



- Consider extending ClassCodeVisitorSupport but also consider extending AbstractASTTransformation or using ClassCodeExpressionTransformer

Writing a Local AST Transform…

- Create an Annotation

```
class MainTransformation{}

import org.codehaus.groovy.transform.GroovyASTTransformationClass
import java.lang.annotation.*

@Retention(RetentionPolicy.SOURCE)
@Target([ElementType.METHOD])
@GroovyASTTransformationClass(classes = [MainTransformation])
public @interface Main {}
```

  - Annotated with @GroovyASTTransformationClass

…Writing a Local AST Transform…

- Write your transform class

```
@GroovyASTTransformation(phase = CompilePhase.INSTRUCTION_SELECTION)
class MainTransformation implements ASTTransformation {
   private static final ClassNode[] NO_EXCEPTIONS =
      ClassNode.EMPTY_ARRAY
   private static final ClassNode STRING_ARRAY =
      ClassHelper.STRING_TYPE.makeArray()

   void visit(ASTNode[] astNodes, SourceUnit sourceUnit) {
      // use guard clauses as a form of defensive programming
      if (!astNodes) return
      if (!astNodes[0] || !astNodes[1]) return
      if (!(astNodes[0] instanceof AnnotationNode)) return
      if (astNodes[0].classNode?.name != Main.class.name) return
      if (!(astNodes[1] instanceof MethodNode)) return
      ...
```

…Writing a Local AST Transform…

```
    ...
    MethodNode annotatedMethod = astNodes[1]
    ClassNode declaringClass = annotatedMethod.declaringClass
    def callMethod = callX(ctorX(declaringClass), annotatedMethod.name)
    Statement body = block(stmt(callMethod))
    def visibility = ACC_STATIC | ACC_PUBLIC
    def parameters = params(param(STRING_ARRAY, 'args'))
    declaringClass.addMethod('main', visibility, VOID_TYPE,
                             parameters, NO_EXCEPTIONS, body)
  }
}
```

…Writing a Local AST Transform

- Use the transform

```
class Greeter {
    @Main
    def greet() {
        println "Hello from the greet() method!"
    }
}
```

…Writing a Local AST Transform

- Use the transform

```
class Greeter {
    @Main
    def g
        p
    }
}
```

```
new GroovyShell(getClass().classLoader).evaluate
'''
class Greeter {
    @Main
    def greet() {
        println "Hello from the greet() method!"
    }
}
'''
```

Creating AST…

- By hand (raw)

```
import org.codehaus.groovy.ast.*
import org.codehaus.groovy.ast.stmt.*
import org.codehaus.groovy.ast.expr.*

new ReturnStatement(
        new ConstructorCallExpression(
                ClassHelper.make(Date),
                ArgumentListExpression.EMPTY_ARGUMENTS
        )
)
```

  - Verbose, full IDE supported

…Creating AST…

- By hand (with helper utility methods)

```
import static org.codehaus.groovy.ast.tools.GeneralUtils.*
import static org.codehaus.groovy.ast.ClassHelper.*

returnS(ctorX(make(Date)))
```

  - Concise, not everything has concise form (yet)

…Creating AST…

- With ASTBuilder (from a specification/DSL)

```
import org.codehaus.groovy.ast.builder.AstBuilder

def ast = new AstBuilder().buildFromSpec {
    returnStatement {
        constructorCall(Date) {
            argumentList {}
        }
    }
}
```

  - Requires AST knowledge, limited IDE support

…Creating AST…

- With ASTBuilder (from a String)

```
import org.codehaus.groovy.ast.builder.AstBuilder

def ast = new AstBuilder().buildFromString('new Date()')
```

  - Concise, intuitive, can't create everything, limited IDE support

…Creating AST…

- With ASTBuilder (from code)

```
import org.codehaus.groovy.ast.builder.AstBuilder

def ast = new AstBuilder().buildFromCode {
    new Date()
}
```

  - Clear, concise, some entities cannot be created, IDE assistance

…Creating AST

- ASTBuilder limitations
  - Great for prototyping, not always suitable for production transforms
  - Groovy technology, can be slow, subject to global transforms
  - Sometimes wasteful, e.g. you might need to create more than you need such as creating a whole class to then pull out one method
  - Some flavors don't support arbitrary node types, don't make it easy to handle interactions with existing class nodes, don't make it easy to support redirects or generics, nor allow you to properly set the line/column numbers resulting in difficult to debug AST transform and cryptic compilation errors

Feature Interactions

- Consider Groovy's @ToString annotation transform which runs at the end of the Canonicalization phase
- Now suppose we want to create a @Trace annotation transform which when placed on a class will "instrument" each method with "trace" println statements, e.g. this:

```
def setX(x) {
  this.x = x
}
```

becomes:

```
def setX(x) {
    println "setX begin"
    this.x = x
    println "setX end"
}
```

- What behaviour should I expect calling toString() if @Trace runs at the end of Semantic Analysis? Canonicalization? Instruction Selection?

Testing AST Transforms

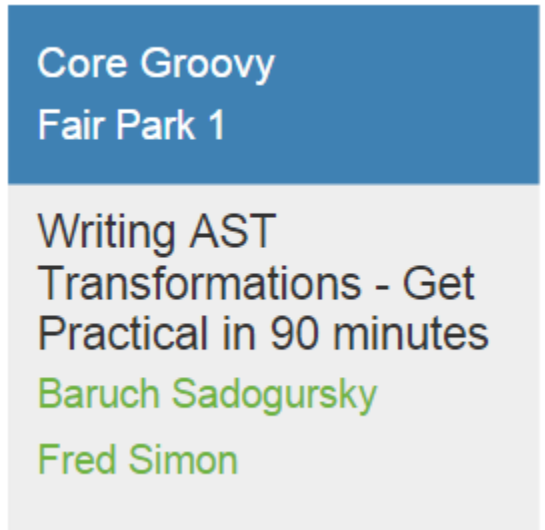
- Consider using ASTTest
- Test both the AST tree and the end-to-end behavior
- Consider writing defensive guards
- Use GroovyConsole

Design Considerations

- Don't reuse ClassNodes
- Compile-time vs Runtime trade-offs and typing
- Feature Interactions/Fragility
- Beware Complexity
- Risks of introducing bugs
- Avoid global transforms (unless needed)
- GroovyConsole is your friend (AST/bytecode)
- Use addError for errors
- Retain line/column number information when transforming
- Watch variable scoping

Further Information

- Documentation
  - http://beta.groovy-lang.org/docs/groovy-2.4.0-SNAPSHOT/html/documentation/#_compile_time_metaprogramming
- Macro Groovy
  - https://github.com/bsideup/MacroGroovy
  - https://github.com/bsideup/groovy-macro-methods
- AST Workshop
  - http://melix.github.io/ast-workshop/
- Other talks



Further Information: Groovy in Action