OCI | WE ARE SOFTWARE ENGINEERS.

APACHECON North America

Groovy Roadmap

Dr Paul King
OCI Groovy Lead
@paulk_asert
https://speakerdeck.com/paulk/groovy-roadmap
https://github.com/paulk-asert/upcoming-groovy

objectcomputing.com

WE ARE SOFTWARE ENGINEERS.

We deliver mission-critical software solutions that accelerate innovation within your organization and stand up to the evolving demands of your business.

OCI | WE ARE SOFTWARE ENGINEERS.

- 160+ engineers
- Home of Grails & Micronaut
- Friend of Groovy
- Global Footprint

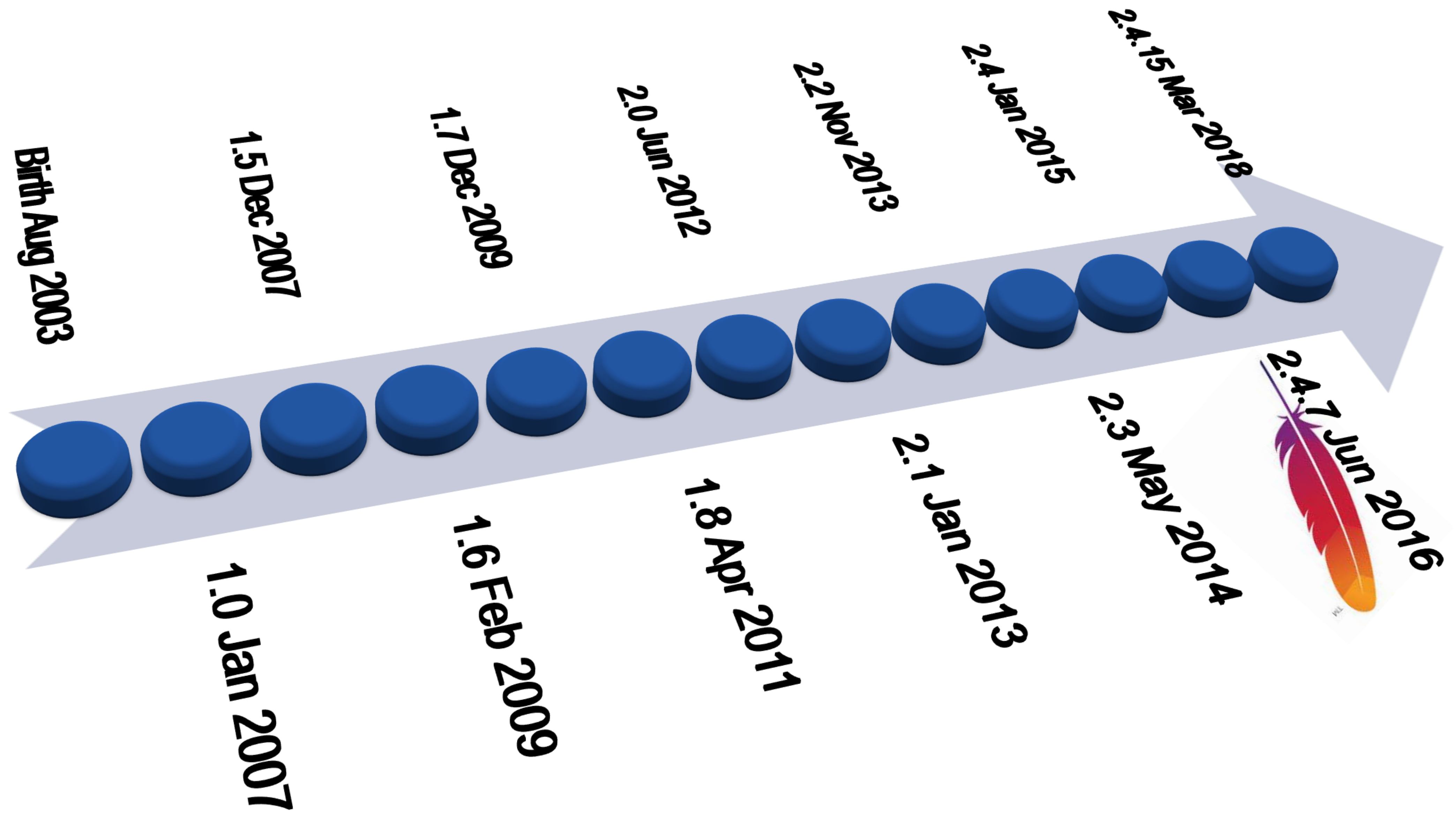
Groovy by the Numbers

- 2.4 in maintenance, 2.5 current, 3.0 in development
- Popular and growing
  2016: 23M
  2017: 50M
  May/Jun/Jul 2018: 27M+
- 18 releases and 40+ new contributors in last 12 months
- Could do with even more contributors! ☺

Groovy Roadmap

- **Groovy 2.5**
  - 2.5.2 released, 2.5.3 soon*
  - Macros, AST transformation improvements, various misc features
  - JDK 7 minimum, runs on JDK 9/10/11* with warnings
- **Groovy 3.0**
  - Alphas out now, betas by end 2018/RCs early 2019
  - Parrot parser, various misc features
  - JDK 8 minimum (3.0), address most JDK 9/10/11/12 issues

But first, how did we get here?



Some common languages when Groovy was born

Dynamic

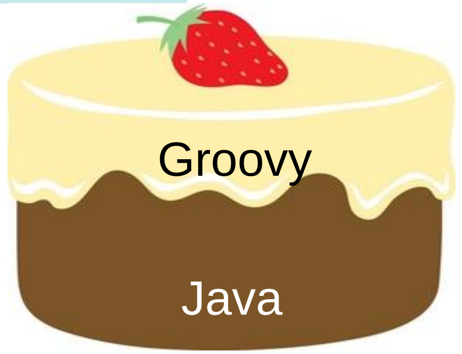
- Ruby
- JavaScript
- Smalltalk
- Python

Static

- Haskell
- Scala
- C#
- Java

Some common languages when Groovy was born

Dynamic	*Static*
Ruby	Haskell
JavaScript	Scala
Smalltalk	C#
Python	Java
Groovy	



Brief history

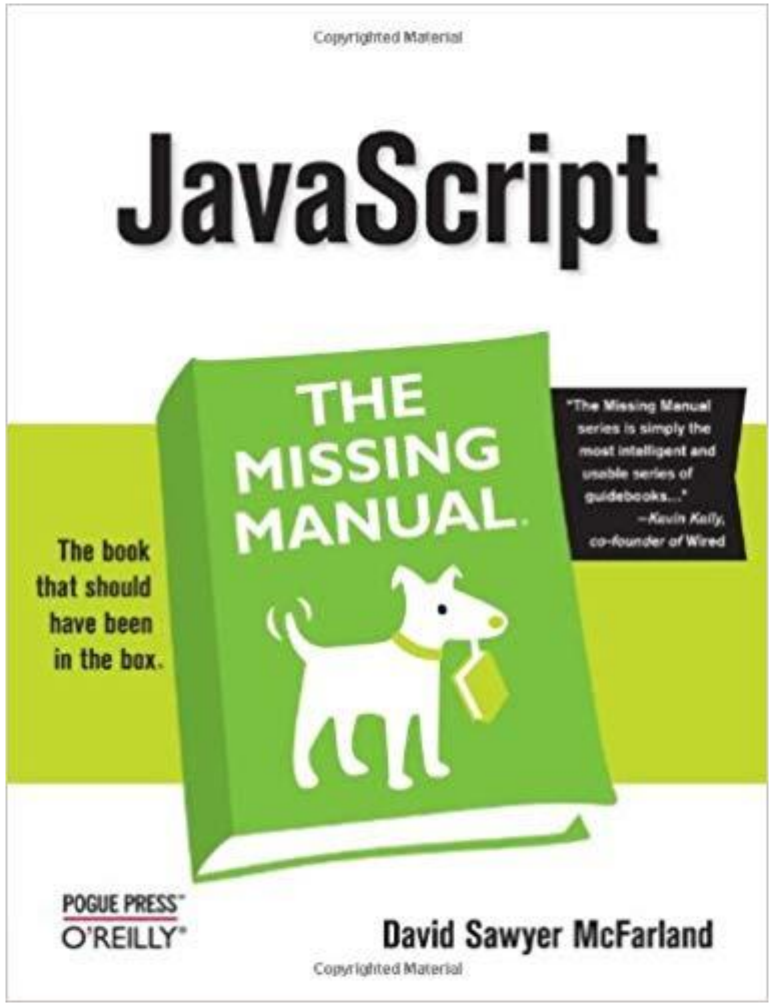
❖ **Groovy 1.0**

Most of Java plus:

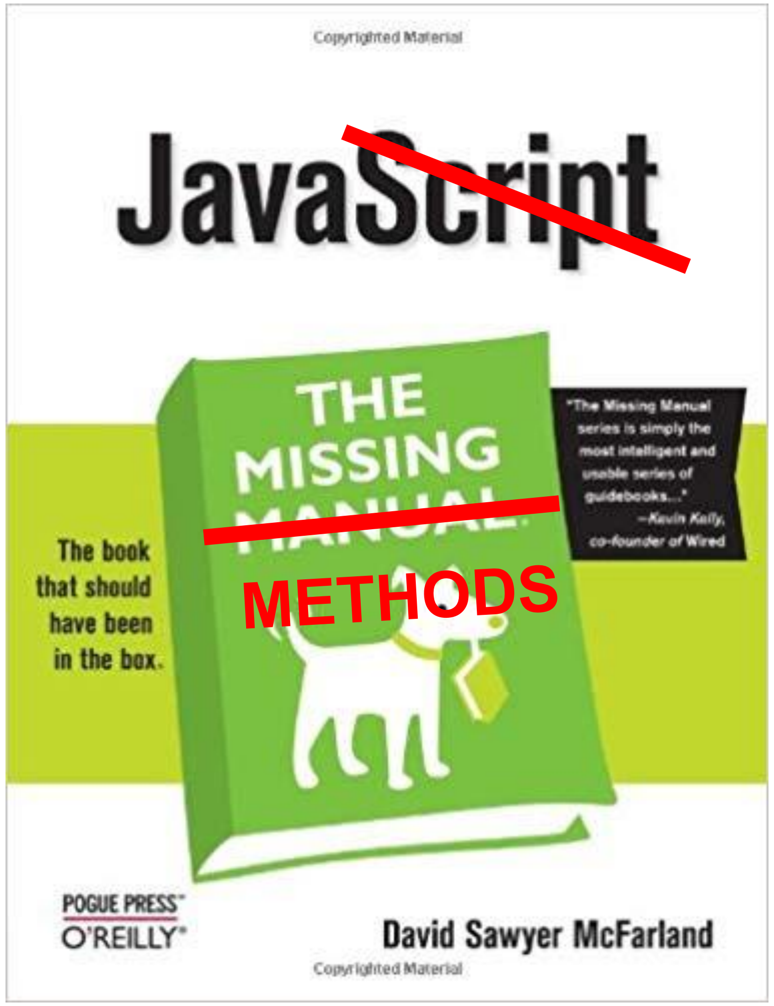
- Closures, scripts, builders, GStrings, named parameters
- Properties, regex, operator overloading, GPath expressions
- Runtime metaprogramming, optional typing, ranges
- GDK (430 methods)

32 committers

GDK – Groovy Development Kit


Copyrighted Material
JavaScript
THE MISSING MANUAL.
"The Missing Manual series is simply the most intelligent and usable series of guidebooks..." —Kevin Kelly, co-founder of Wired
The book that should have been in the box.
POGUE PRESS
O'REILLY
David Sawyer McFarland
Copyrighted Material

GDK – Groovy Development Kit


Copyrighted Material
~~JavaScript~~
THE
MISSING
~~MANUAL~~
METHODS
The book that should have been in the box.
"The Missing Manual series is simply the most intelligent and usable series of guidebooks..." —Kevin Kelly, co-founder of Wired
POGUE PRESS
O'REILLY
David Sawyer McFarland
Copyrighted Material

GDK – Groovy Development Kit

```
assert new URL('http://groovy.apache.org').text.contains(‘Getting involved')
```

GDK – Groovy Development Kit

```
assert new URL('http://groovy.apache.org').text.contains(‘Getting involved')
```

```
def gh = System.getProperty('groovy.home')
```

```
new File(gh).eachFileRecurse{
    if (it.name == 'team-list.html') assert it.text =~ /mcwhirter/
}
```

GDK – Groovy Development Kit

```
assert new URL('http://groovy.apache.org').text.contains(‘Getting involved')
```

```
def gh = System.getProperty('groovy.home')
```

```
new File(gh).eachFileRecurse{
    if (it.name == 'team-list.html') assert it.text =~ /mcwhirter/
}
```

```
def p = "find $gh -name *team* -print".execute()
p.waitFor()
p = "grep -s mcwhirter ${p.text.trim()}".execute()
p.waitFor()
assert !p.exitValue()
```

GDK – Groovy Development Kit

```
assert new URL('http://groovy.apache.org').text.contains('Getting involved')
```

```
def gh = System.getProperty('groovy.home')
```

```
new File(gh).eachFileRecurse{
    if (it.name == 'team-list.html') assert it.text =~ /mcwhirter/
}
```

```
def p = "find $gh -name *team* -print".execute()
p.waitFor(
p = "grep -
p.waitFor(
assert !p.e
```

```
def found = new AntBuilder().fileScanner {
    fileset(dir:gh, casesensitive:false) {
        include(name:'**/team-list.html')
        containsregexp(expression: /mcwhirter/)
    }
}
assert found
```

Runtime metaprogramming

```
class Bar {
    String name() { "Bar is here" }
    def invokeMethod(String name, args) {
        metaClass.invokeMethod(this, name.toLowerCase(), args)
    }
}

def bar  = new Bar()
println bar.name()
println bar.NAME()
```

Runtime metaprogramming

```
class Foo {
    String name() { "Foo is here" }
}
```

```
class LowerMetaClass extends DelegatingMetaClass  {
    LowerMetaClass(Class clazz) { super(clazz) }
    def invokeMethod(receiver, String name, Object[] args) {
        super.invokeMethod(receiver, name.toLowerCase(), args)
    }
}
```

```
def mc = new LowerMetaClass(Foo)
mc.initialize()
foo = new Foo()
foo.setMetaClass(mc)
println foo.name()
println foo.NAME()
```

Extensibility

GDK, metaprogramming,
operator overloading:

- Let the Groovy team add bells and whistles to the language
- Allow you to do the same

Brief history

❖ **Groovy 1.5**

Additions:

- Java 5 (annotations, generics, enums, varargs, static imports)
- Elvis operator
- Metaprogramming improvements
- GDK (630 methods)

18 committers

Static imports

```
import static java.lang.Math.abs
import static java.lang.Math.PI as π
import static java.lang.Math.cos as cosine
import static java.lang.Math.sin as sine

assert sine(π / 6) + cosine(π / 3) == abs(-1)
```

Brief history

❖ **Groovy 1.6**

Additions:

- Multi-assignments
- Metaprogramming improvements
- AST transformations (10 bundled)
- @Grab
- GDK (790 methods)

21 contributors

Multi-assignment

```
def (len, angle) = cartesianToPolar (x, y)
def (lat, long) = geocode ("Paris, France")
def (_, month, year) = "18th June 2009".split()
```

@Grab

```
@Grab('org.apache.opennlp:opennlp-tools:1.9.0')
import opennlp.tools.langdetect.*

def base = 'http://apache.forsale.plus/opennlp/models'
def url = "$base/langdetect/1.8.3/langdetect-183.bin"
def model = new LanguageDetectorModel(new URL(url))
def detector = new LanguageDetectorME(model)
def best = detector.predictLanguage('Bienvenue à Montréal’)

assert best.lang == 'fra'
println best.confidence
```

```
@Grab('org.apache.commons:commons-lang3:3.8.1')
import static org.apache.commons.lang3.SystemUtils.isJavaVersionAtLeast as atLeast
import static org.apache.commons.lang3.JavaVersion.JAVA_1_8 as Java8
import java.util.stream.IntStream

println atLeast(Java8) ?
        IntStream.range(1, 5).reduce{ a, b -> a + b }.asInt :
        (1..<5).sum()
```

Apache Commons Math RealMatrix

```java
import org.apache.commons.math3.linear.*;

public class MatrixMain {
    public static void main(String[] args) {
        double[][] matrixData = { {1d,2d,3d}, {2d,5d,3d}};
        RealMatrix m = MatrixUtils.createRealMatrix(matrixData);

        double[][] matrixData2 = { {1d,2d}, {2d,5d}, {1d, 7d}};
        RealMatrix n = new Array2DRowRealMatrix(matrixData2);

        RealMatrix o = m.multiply(n);

        // Invert p, using LU decomposition
        RealMatrix oInverse = new LUDecomposition(o).getSolver().getInverse();

        RealMatrix p = oInverse.scalarAdd(1d).scalarMultiply(2d);

        RealMatrix q = o.add(p.power(2));

        System.out.println(q);
    }
}
```

Apache Commons™
http://commons.apache.org/

ExpandoMetaClass DSL

```
@Grab('org.apache.commons:commons-math3:3.6.1')
import org.apache.commons.math3.linear.*

RealMatrix.metaClass {
    plus << { RealMatrix ma -> delegate.add(ma) }
    plus << { double d -> delegate.scalarAdd(d) }
    multiply { double d -> delegate.scalarMultiply(d) }
    bitwiseNegate { -> new LUDecomposition(delegate).solver.inverse }
}
MatrixUtils.metaClass.static.create << { List[] l ->
    MatrixUtils.createRealMatrix(l as double[][]) }
```

ExpandoMetaClass DSL

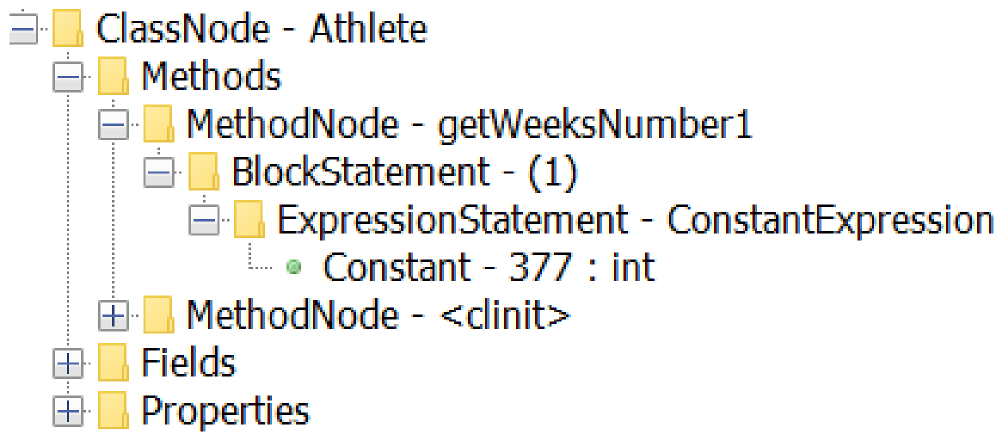
```
@Grab('org.apache.commons:commons-math3:3.6.1')
import org.apache.commons.math3.linear.*

RealMatrix.metaClass {
    plus << { RealMatrix ma -> delegate.add(ma) }
    plus << { double d -> delegate.scalarAdd(d) }
    multiply { c
    bitwiseNeg
}
MatrixUtils.m
    MatrixUtils
```

```
def m = MatrixUtils.create([1d,2d,3d], [2d,5d,3d])
def n = MatrixUtils.create([1d,2d], [2d,5d], [1d, 7d])
def o = m * n
def p = (~o + 1) * 2
def q = o + p ** 2
println q
```

Groovy compilation process

- Multiple phases
- Skeletal AST

```
class Athlete {
    String name, nationality
    int getWeeksNumber1() {
        377
    }
}

new Athlete(name: 'Steffi Graf',
        nationality: 'German')
```

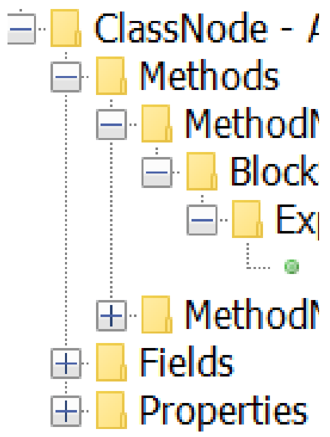


```
ClassNode - Athlete
  Methods
    MethodNode - getWeeksNumber1
      BlockStatement - (1)
        ExpressionStatement - ConstantExpression
          Constant - 377 : int
    MethodNode - <clinit>
  Fields
  Properties
```

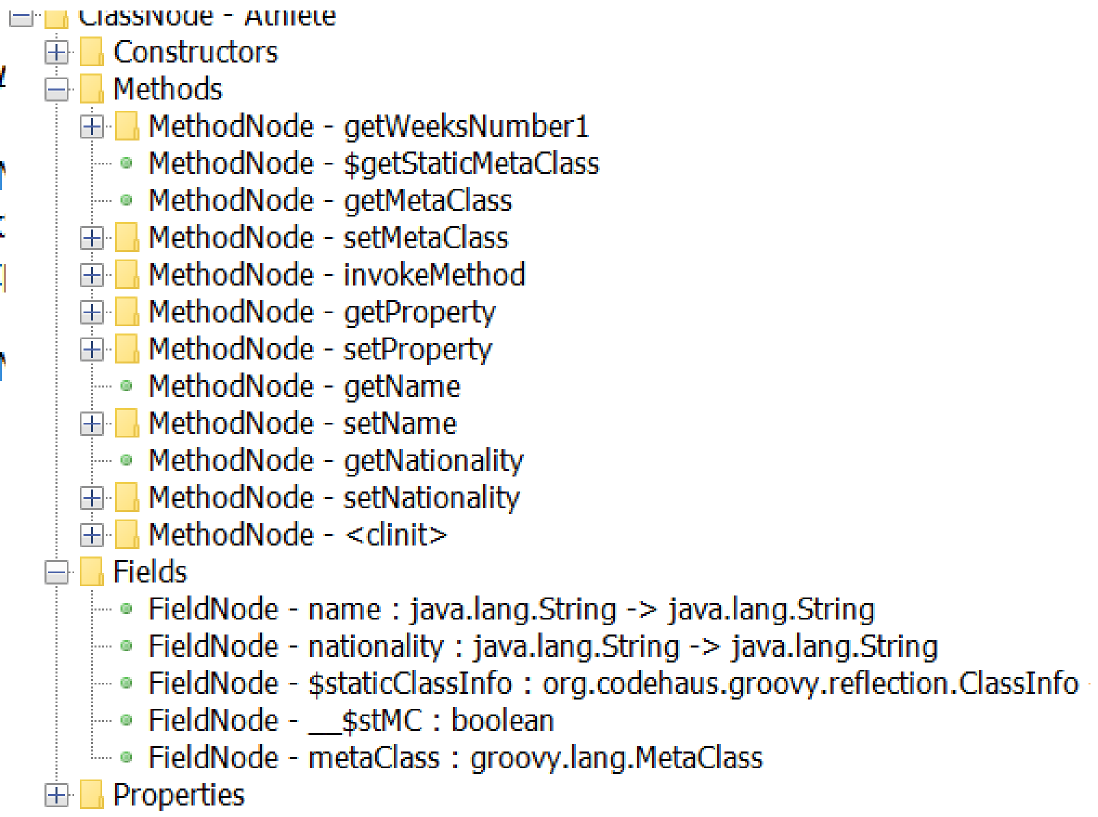
Groovy compilation process

- Multiple phases
- Skeletal AST => Completely resolved enriched AST
- Output bytecode

```
class Athlete {
    String name, nationality
    int getWeeksNumber1() {
        377
    }
}

new Athlete(name: 'Steffi Graf',
        nationality: 'German')
```



```
ClassNode - 
    Methods
        MethodN
            Block
                Ex
        MethodN
    Fields
    Properties
```

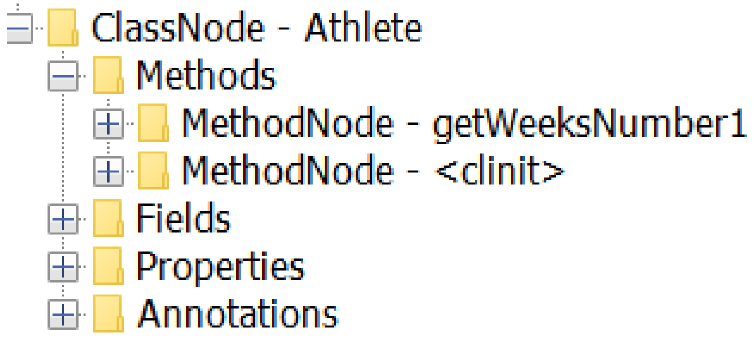


```
ClassNode - Athlete
    Constructors
    Methods
        MethodNode - getWeeksNumber1
        MethodNode - $getStaticMetaClass
        MethodNode - getMetaClass
        MethodNode - setMetaClass
        MethodNode - invokeMethod
        MethodNode - getProperty
        MethodNode - setProperty
        MethodNode - getName
        MethodNode - setName
        MethodNode - getNationality
        MethodNode - setNationality
        MethodNode - <clinit>
    Fields
        FieldNode - name : java.lang.String -> java.lang.String
        FieldNode - nationality : java.lang.String -> java.lang.String
        FieldNode - $staticClassInfo : org.codehaus.groovy.reflection.ClassInfo
        FieldNode - __$stMC : boolean
        FieldNode - metaClass : groovy.lang.MetaClass
    Properties
```

Compile-time metaprogramming: AST transformations

- Global transforms
  - run for all source files
- Local transforms
  - annotations target where transform will be applied
- Manipulate the AST

```
@ToString
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
}

new Athlete(name: 'Steffi Graf',
        nationality: 'German')
```

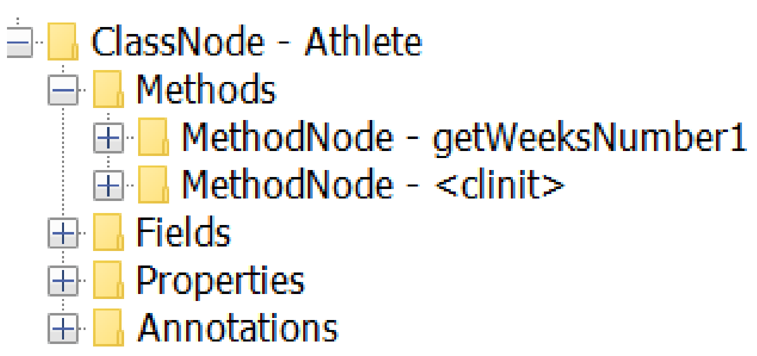


```
ClassNode - Athlete
  Methods
    MethodNode - getWeeksNumber1
    MethodNode - <clinit>
  Fields
  Properties
  Annotations
```

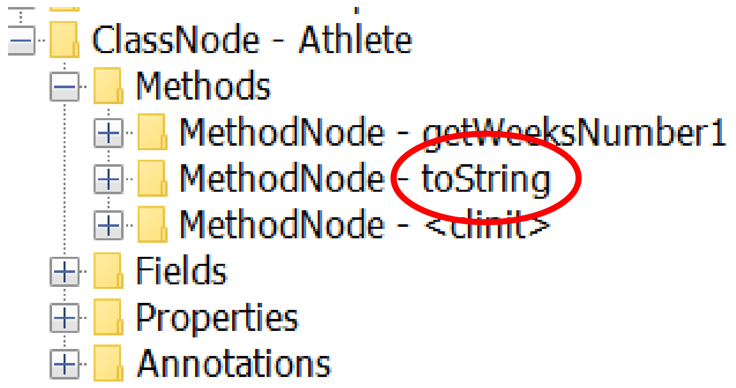
Compile-time metaprogramming: AST transformations

- Global transforms
  - run for all source files
- Local transforms
  - annotations target where transform will be applied
- Manipulate the AST

```
@ToString
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
}

new Athlete(name: 'Steffi Graf',
        nationality: 'German')
```



```
ClassNode - Athlete
  Methods
    MethodNode - getWeeksNumber1
    MethodNode - <clinit>
  Fields
  Properties
  Annotations
```

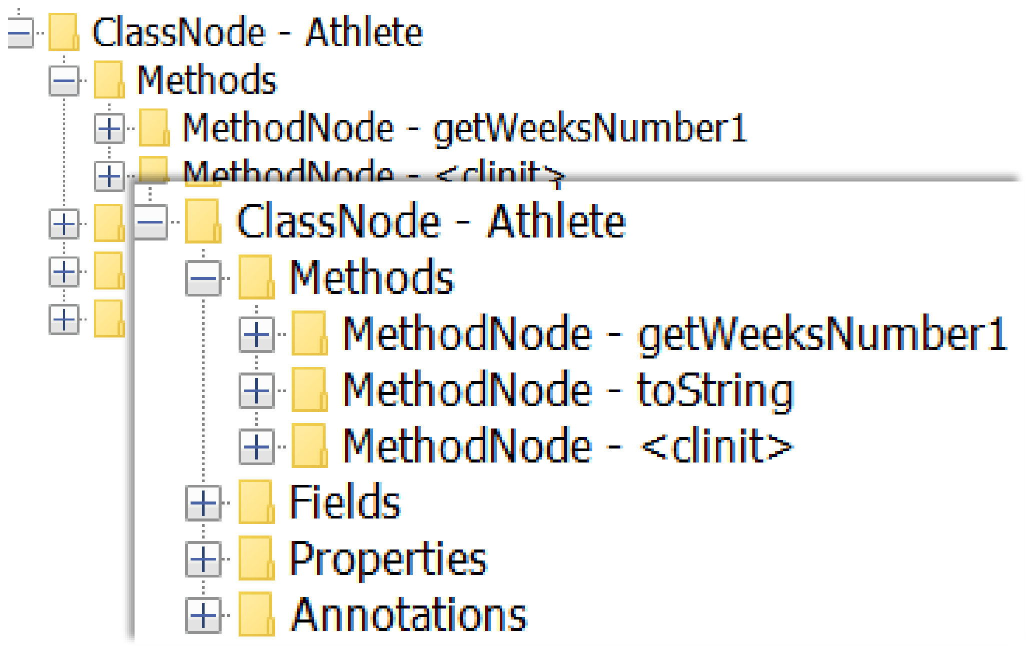


```
ClassNode - Athlete
  Methods
    MethodNode - getWeeksNumber1
    MethodNode - toString
    MethodNode - <clinit>
  Fields
  Properties
  Annotations
```

Compile-time metaprogramming: AST transformations

```
@ToString
class Athlete {
    Strin
    int g
}

new At
     n
```

```
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
    String toString() {
        def sb = new StringBuilder()
        sb << 'Athlete('
        sb << name
        sb << ', '
        sb << nationality
        sb << ')'
        return sb.toString()
    }
}
```



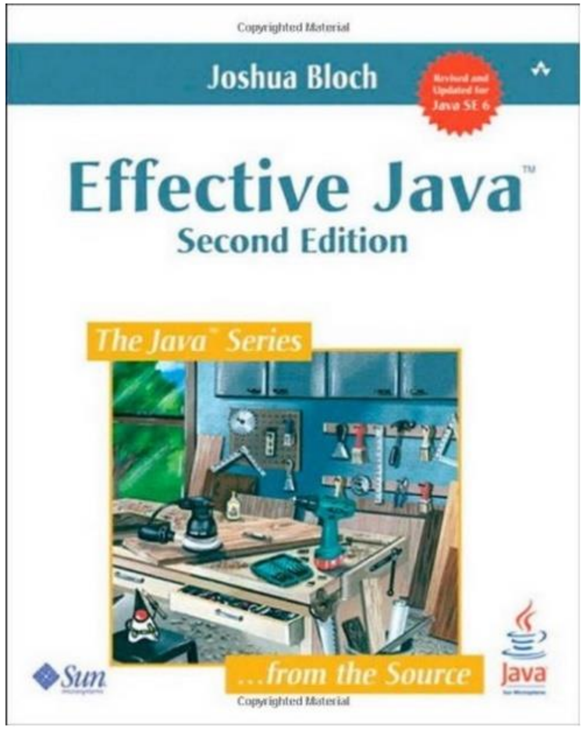
```
ClassNode - Athlete
  Methods
    MethodNode - getWeeksNumber1
    MethodNode - <clinit>
```

```
ClassNode - Athlete
  Methods
    MethodNode - getWeeksNumber1
    MethodNode - toString
    MethodNode - <clinit>
  Fields
  Properties
  Annotations
```

Immutable Classes

Some Rules

- Don’t provide mutators
- Ensure that no methods can be overridden
  - Easiest to make the class final
  - Or use static factories & non-public constructors
- Make all fields final
- Make all fields private
  - Avoid even public immutable constants
- Ensure exclusive access to any mutable components
  - Don’t leak internal references
  - Defensive copying in and out
- Optionally provide *equals* and *hashCode* methods
- Optionally provide *toString* method



@Immutable…

Java Immutable Class

- As per Joshua Bloch Effective Java

```java
public final class Person {
    private final String first;
    private final String last;

    public String getFirst() {
        return first;
    }

    public String getLast() {
        return last;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((first == null)
            ? 0 : first.hashCode());
        result = prime * result + ((last == null)
            ? 0 : last.hashCode());
        return result;
    }

    public Person(String first, String last) {
        this.first = first;
        this.last = last;
    }
    // ...
```

```java
    // ...
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Person other = (Person) obj;
        if (first == null) {
            if (other.first != null)
                return false;
        } else if (!first.equals(other.first))
            return false;
        if (last == null) {
            if (other.last != null)
                return false;
        } else if (!last.equals(other.last))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "Person(first:" + first
            + ", last:" + last + ")";
    }

}
```

…@Immutable…

boilerplate

Java Immutable Class

- As per Joshua Bloch Effective Java

```java
public final class Person {
    private final String first;
    private final String last;

    public String getFirst() {
        return first;
    }

    public String getLast() {
        return last;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((first == null)
            ? 0 : first.hashCode());
        result = prime * result + ((last == null)
            ? 0 : last.hashCode());
        return result;
    }

    public Person(String first, String last) {
        this.first = first;
        this.last = last;
    }
    // ...
```

```java
    // ...
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Person other = (Person) obj;
        if (first == null) {
            if (other.first != null)
                return false;
        } else if (!first.equals(other.first))
            return false;
        if (last == null) {
            if (other.last != null)
                return false;
        } else if (!last.equals(other.last))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "Person(first:" + first
            + ", last:" + last + ")";
    }

}
```

…@Immutable

```
@Immutable class Person {
    String first, last
}
```

@Lazy

```
class Resource{}    // expensive resource
```

```
def res1 = new Resource()

@Lazy res2 = new Resource()

@Lazy static res3 = { new Resource() }()

@Lazy volatile Resource res4

@Lazy(soft=true) volatile Resource res5
```

@Lazy

```
class Resource{}     // expensive resource
```

```
def res1 = new Resource()

@Lazy res2 = new Resource()

@Lazy static res3 = { new Resource() }()

@Lazy volatile Resource res4

@Lazy(soft=true) volatile Resource res5
```

Eager

@Lazy

```
class Resource{}      // expensive resource
```

```
def res1 = new Resource()

@Lazy res2 = new Resource()

@Lazy static res3 = { new Resource() }()

@Lazy volatile Resource res4

@Lazy(soft=true) volatile Resource res5
```

On first use
but not
threadsafe

@Lazy

```
class Resource{}     // expensive resource
```

```
def res1 = new Resource()

@Lazy res2 = new Resource()

@Lazy static res3 = { new Resource() }()

@Lazy volatile Resource res4

@Lazy(soft=true) volatile Resource res5
```

Class holder idiom,
Complex creation

@Lazy

```
class Resource{}      // expensive resource
```

```
def res1 = new Resource()

@Lazy res2 = new Resource()

@Lazy static res3 = { new Resource() }()

@Lazy volatile Resource res4

@Lazy(soft=true) volatile Resource res5
```

Double checked locking, Auto creation.

@Lazy

```
class Resource{}      // expensive resource
```

```
def res1 = new Resource()

@Lazy res2 = new Resource()

@Lazy static res3 = { new Resource() }()

@Lazy volatile Resource res4

@Lazy(soft=true) volatile Resource res5
```

As above but with soft reference.

Brief history

❖ **Groovy 1.8**

Additions:

- Functional improvements: closures as annotation attributes, closure composition, memoization, partial application, trampoline
- Command chains
- GDK (1100 methods)
- 33 AST transforms

11 contributors

Command chains

```
turn left then right
move forward at 3.km/h
take 2.pills of chloroquinine after 6.hours
paint wall with red, green and yellow
check that: margarita tastes good
given { } when { } then { }
select all unique() from names
take 3 cookies
```

Command chains

```
turn left then right
move forward at 3.km/h
take 2.pills of chloroquinine after 6.hours
paint wall with red, green and yellow
check that: margarita tastes good
given { } when { } then { }
select all unique() from names
take 3 cookies
```

```
turn(left).then(right)
move(forward).at(3.getKm().div(h))
take(2.pills).of(chloroquinine).after(6.hours)
paint(wall).with(red, green).and(yellow)
check(that: margarita).tastes(good)
given({}).when({}).then({})
select(all).unique().from(names)
take(3).cookies
```

Command chains

```
// Japanese DSL using GEP3 rules
Object.metaClass.を =
        Object.metaClass.の = { clos -> clos(delegate) }

まず = { it }
表示する = { println it }
平方根 = { Math.sqrt(it) }

まず 100 の 平方根 を 表示する  // First, show the square root of 100
// => 10.0
```

Brief history

❖ **Groovy 2.0**

Additions:

- Static nature to a dynamic language @TypeChecked, @CompileStatic, flow typing, GDK extension methods became static aware
- JDK 7 related changes: project coin
- Modular jars
- Performance
- GDK (1130 methods)
- 33 AST transforms

28 contributors

Runtime type checking with optional typing

```
def first = 'John'
String last = 'Smith'
assert "${first.toLowerCase()} ${last.toUpperCase()}" == 'john SMITH'
```

Static type checking

```
def first = 'John'
String last = 'Smith'
assert "${first.toLowerCase()} ${last.toUpperCase()}" == 'john SMITH'
```

```
import groovy.transform.TypeChecked

@TypeChecked
void assertName() {
    def first = 'John'
    String last = 'Smith'
    assert "${first.toLowerCase()} ${last.toUpperCase()}" == 'john SMITH'
}
```

Type checking with static compilation bytecode

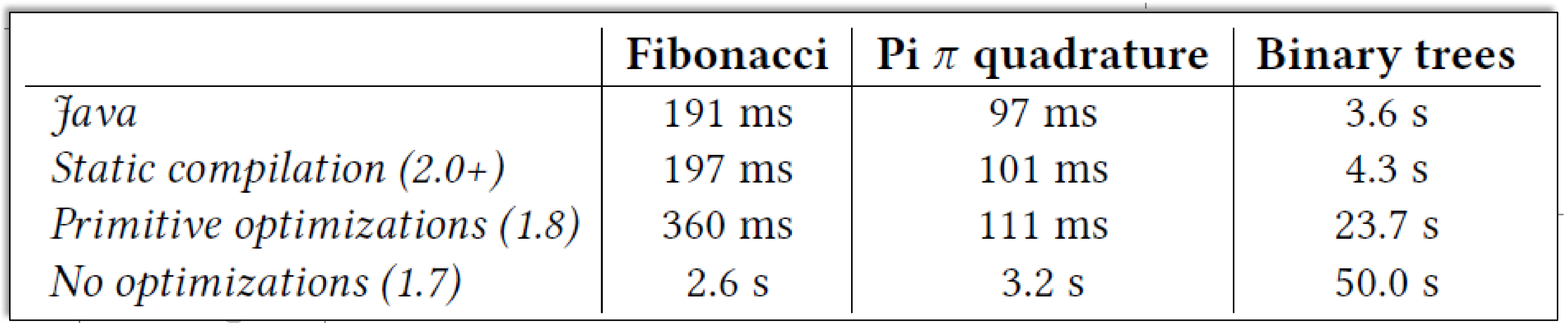
```
def first = 'John'
String last = 'Smith'
assert "${first.toLowerCase()} ${last.toUpperCase()}" == 'john SMITH'
```

```
import groovy.transform.TypeChecked

@TypeChecked
void assertName() {
    def first = 'John'
    String last
    assert "${f
}
```

```
import groovy.transform.CompileStatic

@CompileStatic
void assertName() {
    def first = 'John'
    String last = 'Smith'
    assert "${first.toLowerCase()} ${last.toUpperCase()}" == 'john SMITH'
}
```

@CompileStatic

```
def first = 'John'
String last = 'Smith'
assert "${first.toLowerCase()} ${last.toUpperCase()}" == 'john SMITH'
```



	Fibonacci	Pi π quadrature	Binary trees
Java	191 ms	97 ms	3.6 s
Static compilation (2.0+)	197 ms	101 ms	4.3 s
Primitive optimizations (1.8)	360 ms	111 ms	23.7 s
No optimizations (1.7)	2.6 s	3.2 s	50.0 s

```
    assert "${f
}

assertName(
```

```
@CompileStatic
void assertName() {
    def first = 'John'
    String last = 'Smith'
    assert "${first.toLowerCase()} ${last.toUpperCase()}" == 'john SMITH'
}

assertName()
```

Flow typing

```
def o = 'foobarbaz'
o = o.toUpperCase()    // String
o = o.size()           // int
o = Math.sqrt(o)       // double
assert o == 3
```

* ignoring code style temporarily

Brief history

❖ Groovy 2.1

Additions:

- Better invoke dynamic support
- Improved type checking
- Type checking extensions
- Meta-annotations (aka annotation collectors)
- GDK (1140 methods)
- 34 AST transforms

21 contributors

Type checking extensions

```
class Bar {
    String name() { "Bar is here" }
    def invokeMethod(String name, args) {
        metaClass.invokeMethod(this, name.toLowerCase(), args)
    }
}
```

Type checking extensions

```
class Bar {
    String name() { "Bar is here" }
    def invokeMethod(String name, args) {
        metaClass.invokeMethod(this, name.toLowerCase(), args)
    }
}
```

```
@TypeChecked
def method() {
    def bar  = new Bar()
    println bar.name()
    println bar.NAME()
}
```

✗

Type checking extensions

```
methodNotFound { receiver, name, argumentList, argTypes, call ->
    def result = null
    withTypeChecker {
        def candidates = findMethod(receiver, name.toLowerCase(), argTypes)
        if (candidates && candidates.size() == 1) {
            result = candidates[0]
        }
    }
    result
}
```

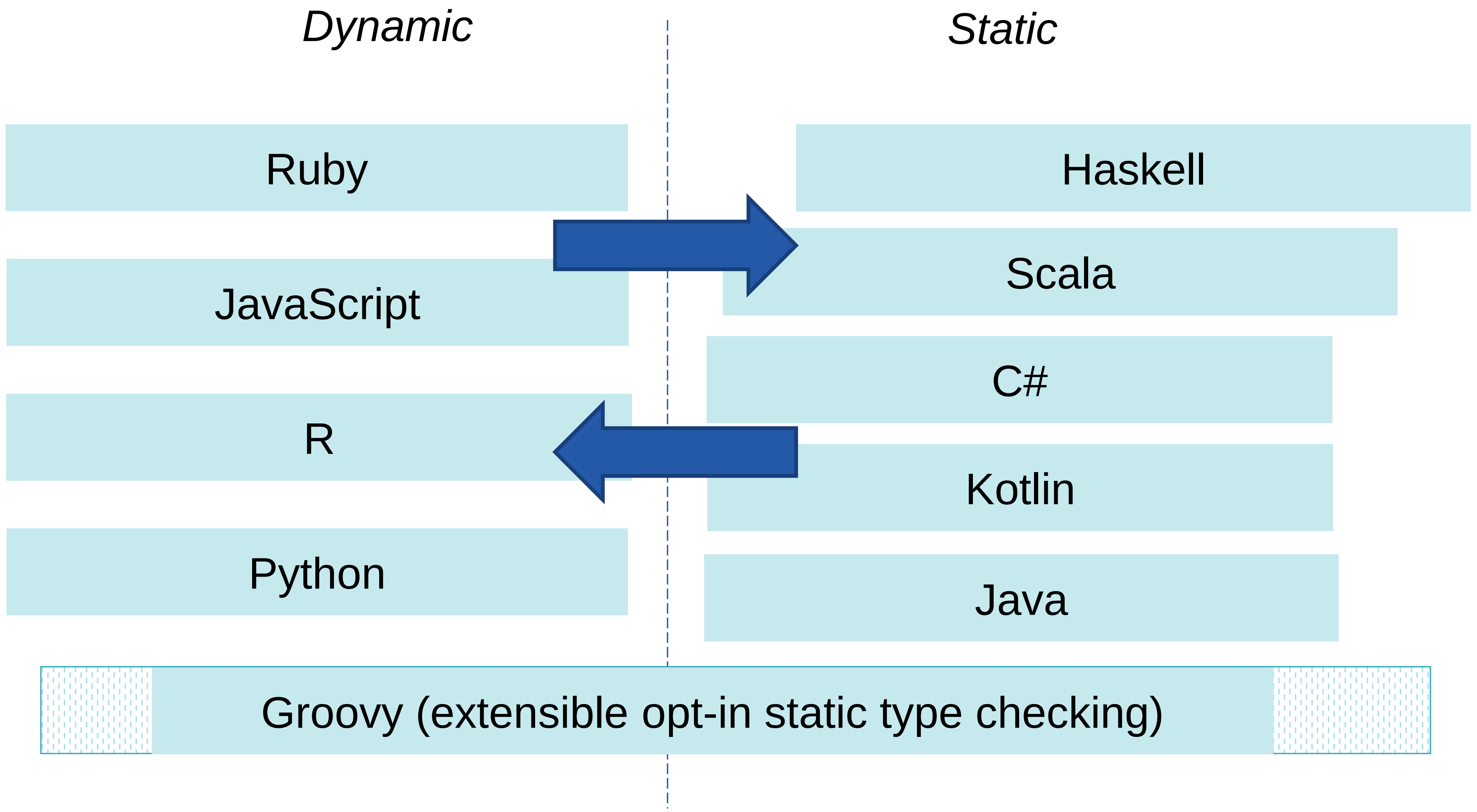
Type checking extensions

```
methodNotFound { receiver, name, argumentList, argTypes, call ->
    def result = null
    withTypeChecker {
        def candidates = findMethod(receiver, name.toLowerCase(), argTypes)
        if (candidates && candidates.size() == 1) {
            result = candidates[0]
        }
    }
    result
}
```

```
@TypeChecked(extensions = 'LowerChecker.groovy')
def method() {
    def bar  = new Bar()
    println bar.name()
    println bar.NAME()
}
```

☑

Changes over time



Brief history

❖ **Groovy 2.2**

Additions:

- SAM coercion
- @Memoized
- GDK (1170 methods)
- 37 AST transforms

46 contributors

SAM Coercion

Without this, some Java expressions would be shorter than the Groovy equivalents!

Brief history

❖ **Groovy 2.3**

Additions:

- JDK8 official support
- Traits
- GDK (1270 methods)
- 43 AST transforms

45 contributors

Traits

```
trait FlyingAbility {
    String fly() { "Flap!" }
}

trait SpeakingAbility {
    String speak() { "Quack!" }
}

class Duck implements FlyingAbility, SpeakingAbility {}

def d = new Duck()
assert d.fly() == "Flap!"
assert d.speak() == "Quack!"
```

Brief history

❖ **Groovy 2.4**

Additions:

- Android support
- Many improvements
- Joined Apache after 2.4.6
- GDK (1350 methods)
- 46 AST transforms

58 contributors

Groovy 2.5 Themes

- **AST Transformation improvements**
- Macros
- Misc improvements
- Runs on JDK 9/10/11*
  - But with benign (but annoying) warnings (* soon)

Groovy 2.5 Themes

- AST Transformation improvements
- **Macros**
- Misc improvements
- Runs on JDK 9/10/11*
  - But with benign (but annoying) warnings (* soon)

Groovy 2.5 Themes

- AST Transformation improvements
- Macros
- **Misc improvements**
- Runs on JDK 9/10/11*
  - But with benign (but annoying) warnings (* soon)

Groovy 2.5 Themes

- AST Transformation improvements
- Macros
- Misc improvements
- **Runs on JDK 9/10/11***
  - But with benign (but annoying) warnings (* soon)

AST Transformations – Groovy 2.4, Groovy 2.5, Groovy 3.0

@ASTTest
@AutoClone
@AutoExternalize
@BaseScript
@Bindable
@Builder
@Canonical
@Category
@CompileDynamic
@CompileStatic
@ConditionalInterrupt
@Delegate
@EqualsAndHashCode
@ExternalizeMethods
@ExternalizeVerifier
@Field

@Grab
- @GrabConfig
- @GrabResolver
- @GrabExclude

@Grapes
@Immutable
@IndexedProperty
@InheritConstructors
@Lazy
Logging:
- @Commons
- @Log
- @Log4j
- @Log4j2
- @Slf4j

@ListenerList
@~~Mixin~~

@Newify
@NotYetImplemented
@PackageScope
@Singleton
@Sortable
@SourceURI
@Synchronized
@TailRecursive
@ThreadInterrupt
@TimedInterrupt
@ToString
@Trait
@TupleConstructor
@TypeChecked
@Vetoable
@WithReadLock
@WithWriteLock

@AutoFinal
@AutoImplement
@ImmutableBase
@ImmutableOptions
@MapConstructor
@NamedDelegate
@NamedParam
@NamedParams
@NamedVariant
@PropertyOptions
@VisibilityOptions

@GroovyDoc

AST Transformations – Groovy 2.4, Groovy 2.5, Groovy 3.0

@ASTTest
@AutoClone
@AutoExternalize
@BaseScript
@Bindable
@Builder
@Canonical
@Category
@CompileDynamic
@CompileStatic
@ConditionalInterrupt
@Delegate
@EqualsAndHashCode
@ExternalizeMethods
@ExternalizeVerifier
@Field

@Grab
- @GrabConfig
- @GrabResolver
- @GrabExclude

@Grapes
@Immutable
@IndexedProperty
@InheritConstructors
@Lazy
Logging:
- @Commons
- @Log
- @Log4j
- @Log4j2
- @Slf4j

@ListenerList
@~~Mixin~~

@Newify
@NotYetImplemented
@PackageScope
@Singleton
@Sortable
@SourceURI
@Synchronized
@TailRecursive
@ThreadInterrupt
@TimedInterrupt
@ToString
@Trait
@TupleConstructor
@TypeChecked
@Vetoable
@WithReadLock
@WithWriteLock

@AutoFinal
@AutoImplement
@ImmutableBase
@ImmutableOptions
@MapConstructor
@NamedDelegate
@NamedParam
@NamedParams
@NamedVariant
@PropertyOptions
@VisibilityOptions

@GroovyDoc

* Improved in 2.5

AST Transformations – Groovy 2.4, Groovy 2.5

Numerous annotations have additional annotation attributes, e.g @TupleConstructor

```
String[] excludes() default {}
String[] includes() default {Undefined.STRING}
boolean includeProperties() default true
boolean includeFields() default false
boolean includeSuperProperties() default false
boolean includeSuperFields() default false
boolean callSuper() default false
boolean force() default false

boolean defaults() default true
boolean useSetters() default false
boolean allNames() default false
boolean allProperties() default false
String visibilityId() default Undefined.STRING
Class pre() default Undefined.CLASS
Class post() default Undefined.CLASS
```

AST Transformations – Groovy 2.5

Some existing annotations totally reworked:

@Canonical and @Immutable are now meta-annotations (annotation collectors)

AST Transforms: @Canonical becomes meta-annotation

@Canonical =>

@ToString, @TupleConstructor, @EqualsAndHashCode

AST Transforms: @Canonical becomes meta-annotation

@Canonical =>

@ToString, @TupleConstructor, @EqualsAndHashCode

```
@AnnotationCollector(
  value=[ToString, TupleConstructor, EqualsAndHashCode],
  mode=AnnotationCollectorMode.PREFER_EXPLICIT_MERGED)
public @interface Canonical { }
```

@Canonical

```
@Canonical(cache = true,
           useSetters = true,
           includeNames = true)
class Point {
    int x, y
}
```

@Canonical

```
@Canonical(cache = true,
           useSetters = true,
           includeNames = true)
class Point {
    int x, y
}
```

```
@ToString(cache = true, includeNames = true)
@TupleConstructor(useSetters = true)
@EqualsAndHashCode(cache = true)
class Point {
    int x, y
}
```

AST Transforms: @Immutable becomes meta-annotation

```
@Immutable
class Point {
    int x, y
}
```

AST Transforms: @Immutable becomes meta-annotation

```
@Immutable
class Point {
    int x, y
}
```

```
@ToString(includeSuperProperties = true, cache = true)
@EqualsAndHashCode(cache = true)
@ImmutableBase
@ImmutableOptions
@PropertyOptions(propertyHandler = ImmutablePropertyHandler)
@TupleConstructor(defaults = false)
@MapConstructor(noArg = true, includeSuperProperties = true, includeFields = true)
@KnownImmutable
class Point {
    int x, y
}
```

AST Transforms: @Immutable enhancements

An immutable class with one constructor making it dependency injection friendly

```
import groovy.transform.*
import groovy.transform.options.*

@ImmutableBase
@PropertyOptions(propertyHandler = ImmutablePropertyHandler)
@Canonical(defaults=false)
class Shopper {
    String first, last
    Date born
    List items
}

println new Shopper('John', 'Smith', new Date(), [])
```

AST Transforms: @Immutable enhancements

JSR-310 classes recognized as immutable

```
java.time.DayOfWeek
java.time.Duration
java.time.Instant
java.time.LocalDate
java.time.LocalDateTime
java.time.LocalTime
java.time.Month
java.time.MonthDay
java.time.OffsetDateTime
java.time.OffsetTime
java.time.Period
java.time.Year
java.time.YearMonth
java.time.ZonedDateTime
java.time.ZoneOffset
java.time.ZoneRegion
// all interfaces from java.time.chrono.*
java.time.format.DecimalStyle
java.time.format.FormatStyle
java.time.format.ResolverStyle
java.time.format.SignStyle
java.time.format.TextStyle
java.time.temporal.IsoFields
java.time.temporal.JulianFields
java.time.temporal.ValueRange
java.time.temporal.WeekFields
```

AST Transforms: @Immutable enhancements

You can write custom property handlers, e.g. to use Guava immutable collections for any collection property

```groovy
import groovy.transform.Immutable
import paulk.transform.construction.GuavaImmutablePropertyHandler
@Immutable(propertyHandler=GuavaImmutablePropertyHandler)
class Person {
    List names = ['John', 'Smith']
    List books = ['GinA', 'ReGinA']
}

['names', 'books'].each {
    println new Person()."$it".dump()
}
//<com.google.common.collect.RegularImmutableList@90b9bd9 array=[John, Smith]>
//<com.google.common.collect.RegularImmutableList@95b86f34 array=[GinA, ReGinA]>
```

AST Transforms: @Immutable handles Optional

```
import groovy.transform.Immutable

@Immutable
class Entertainer {
    String first
    Optional<String> last
}

println new Entertainer('Sonny', Optional.of('Bono'))
println new Entertainer('Cher', Optional.empty())
```

```
Entertainer(Sonny, Optional[Bono])
Entertainer(Cher, Optional.empty)
```

```
@Immutable
class Line {
    Optional<java.awt.Point> origin
}
```

@Immutable processor doesn't know how to handle field ‘origin' of type 'java.util.Optional‘ while compiling class Template…

AST Transforms: property name validation

```
import groovy.transform.ToString

@ToString(excludes = 'first')
class Cyclist {
    String firstName, lastName
}
println new Cyclist(firstName: 'Cadel', lastName: 'Evans')
```

Error during @ToString processing:
'excludes' property 'first' does not exist.

- Transforms check property names and you can call the same methods in your custom transforms

AST Transforms: @TupleConstructor defaults

```groovy
import groovy.transform.TupleConstructor

@TupleConstructor(defaults = false)
class Flight {

    String fromCity, toCity
    Date leaving
}

@TupleConstructor(defaults = true)
class Cruise {

    String fromPort, toPort
    Date leaving
}
```

AST Transforms: @TupleConstructor pre/post

```groovy
import groovy.transform.ToString
import groovy.transform.TupleConstructor

import static groovy.test.GroovyAssert.shouldFail

@ToString
@TupleConstructor(
    pre = { first = first?.toLowerCase(); assert last },
    post = { this.last = first?.toUpperCase() }
)
class Actor {
    String first, last
}

assert new Actor('Johnny', 'Depp').toString() == 'Actor(johnny, JOHNNY)'
shouldFail(AssertionError) {
    println new Actor('Johnny')
}
```

AST Transforms: @TupleConstructor enhancements

Visibility can be specified, also works with MapConstructor and NamedVariant

```
import groovy.transform.*
import static groovy.transform.options.Visibility.PRIVATE

@TupleConstructor
@VisibilityOptions(PRIVATE)
class Person {
    String name
    static makePerson(String first, String last) {
        new Person("$first $last")
    }
}

assert 'Jane Eyre' == Person.makePerson('Jane', 'Eyre').name
def publicCons = Person.constructors
assert publicCons.size() == 0
```

AST Transforms: @MapConstructor

```
import groovy.transform.MapConstructor
import groovy.transform.ToString

@ToString(includeNames = true)
@MapConstructor
class Conference {
    String name
    String city
    Date start
}

println new Conference(
    name: 'Gr8confUS', city: 'Minneapolis', start: new Date() - 2)
println Conference.constructors
```

```
Conference(name:Gr8confUS, city:Minneapolis, start:Wed Jul 26 ...)
[public Conference(java.util.Map)]
```

AST Transforms: @AutoImplement

Designed to complement Groovy’s dynamic creation of “dummy” objects

```
def testEmptyIterator(Iterator it) {
    assert it.toList() == []
}

def emptyIterator = [hasNext: {false}] as Iterator

testEmptyIterator(emptyIterator)

assert emptyIterator.class.name.contains('Proxy')
```

AST Transforms: @AutoImplement

```
@AutoImplement
class MyClass extends AbstractList<String>
              implements Closeable, Iterator<String> { }
```

AST Transforms: @AutoImplement

```
class MyClass extends AbstractList<String> implements Closeable, Iterator<String> {
    String get(int param0) {
        return null
    }

    String next() {
        return null
    }

    boolean hasNext() {
        return false
    }

    void close() throws Exception {
    }

    int size() {
        return 0
    }
}
```

```
@AutoImplement
class MyClass extends AbstractList<String>
                   implements Closeable, Iterator<String> { }
```

AST Transforms: @AutoImplement

```
class MyClass extends AbstractList<String> implements Closeable, Iterator<String> {
    String get(int param0) {
        return null
    }

    String next() {
        return null
    }

    boolean hasNext() {
        return false
    }

    void close() throws Exception {
    }

    int size() {
        return 0
    }
}
```

```
@AutoImplement
class MyClass extends AbstractList<String>
                    implements Closeable, Iterator<String> { }
```

```
def myClass = new MyClass()

testEmptyIterator(myClass)

assert myClass instanceof MyClass
assert Modifier.isAbstract(Iterator.getDeclaredMethod('hasNext').modifiers)
assert !Modifier.isAbstract(MyClass.getDeclaredMethod('hasNext').modifiers)
```

AST Transforms: @AutoImplement

```
@AutoImplement(exception = UncheckedIOException)
class MyWriter extends Writer { }
```

```
@AutoImplement(exception = UnsupportedOperationException,
        message = 'Not supported by MyIterator')
class MyIterator implements Iterator<String> { }
```

```
@AutoImplement(code = { throw new UnsupportedOperationException(
        'Should never be called but was called on ' + new Date()) })
class EmptyIterator implements Iterator<String> {
    boolean hasNext() { false }
}
```

Built-in AST Transformations @AutoFinal

Automatically adds final modifier to constructor and method parameters

```
import groovy.transform.AutoFinal

@AutoFinal
class Animal {
    private String type
    private Date lastFed

    Animal(String type) {
        this.type = type.toUpperCase()
    }

    def feed(String food) {
        lastFed == new Date()
    }
}
```

```
class Zoo {
    private animals = []
    @AutoFinal
    def createZoo(String animal) {
        animals << new Animal(animal)
    }

    def feedAll(String food) {
        animals.each{ it.feed(food) }
    }
}

new Zoo()
```

Built-in AST Transformations @AutoFinal

- Automatically adds final modifier to constructor and method parameters

```
import groovy.transform.AutoFinal

@AutoFinal
class Animal {
    private String type
    private Date lastFed

    Animal(final String type) {
        this.type = type.toUpperCase()
    }

    def feed(final String food) {
        lastFed == new Date()
    }
}
```

```
class Zoo {
    private animals = []
    @AutoFinal
    def createZoo(final String animal) {
        animals << new Animal(animal)
    }

    def feedAll(String food) {
        animals.each{ it.feed(food) }
    }
}

new Zoo()
```

Built-in AST Transformations @Delegate enhancements

@Delegate can be placed on a getter rather than a field

```groovy
class Person {
    String first, last
    @Delegate
    String getFullName() {
        "$first $last"
    }
}

def p = new Person(first: 'John', last: 'Smith')
assert p.equalsIgnoreCase('JOHN smith')
```

@NamedVariant, @NamedParam, @NamedDelegate

```
import groovy.transform.*
import static groovy.transform.options.Visibility.*

class Color {
    private int r, g, b

    @VisibilityOptions(PUBLIC)
    @NamedVariant
    private Color(@NamedParam int r, @NamedParam int g, @NamedParam int b) {
        this.r = r
        this.g = g
        this.b = b
    }
}

def pubCons = Color.constructors
assert pubCons.size() == 1
assert pubCons[0].parameterTypes[0] == Map
```

@NamedVariant, @NamedParam, @NamedDelegate

```
import groovy.transform.*
import static groovy.transform.options.Visibility.*

class Color {
    private int r, g, b

    @VisibilityOptions(PUBLIC)
    @NamedVariant
    private Color(@NamedParam int r, @NamedParam int g, @NamedParam int b) {
        this.r = r
        this.g = g
        this.b = b
    }
}

def pubCons = Color.constru
assert pubCons.size() == 1
assert pubCons[0].parameter
```

```
public Color(@NamedParam(value = 'r', type = int)
             @NamedParam(value = 'g', type = int)
             @NamedParam(value = 'b', type = int)
             Map __namedArgs) {
    this( __namedArgs.r, __namedArgs.g, __namedArgs.b )
    // plus some key value checking
}
```

@NamedVariant, @NamedParam, @NamedDelegate

```
import groovy.transform.*

class Animal {
    String type, name
}

@ToString(includeNames=true)
class Color {
    Integer r, g, b
}

@NamedVariant
String foo(String s1, @NamedParam String s2,
           @NamedDelegate Color shade,
           @NamedDelegate Animal pet) {
    "$s1 $s2 ${pet.type?.toUpperCase()}:$pet.name $shade"
}

def result = foo(s2: 'S2', g: 12, b: 42, r: 12,
        type: 'Dog', name: 'Rover', 'S1')
assert result == 'S1 S2 DOG:Rover Color(r:12, g:12, b:42)'
```

@NamedVariant, @NamedParam, @NamedDelegate

```
import groovy.transform.*

class Animal {
    String type, name
}

@ToSt
class
    I
}

@Name
Strin



    "
}

def result = foo(s2: 'S2', g: 12, b: 42, r: 12,
        type: 'Dog', name: 'Rover', 'S1')
assert result == 'S1 S2 DOG:Rover Color(r:12, g:12, b:42)'
```

```
String foo(@NamedParam(value = 's2', type = String)
           @NamedParam(value = 'r', type = Integer)
           @NamedParam(value = 'g', type = Integer)
           @NamedParam(value = 'b', type = Integer)
           @NamedParam(value = 'type', type = String)
           @NamedParam(value = 'name', type = String)
           Map __namedArgs, String s1) {
    // some key validation code ...
    return this.foo(s1, __namedArgs.s2,
            ['r': __namedArgs.r, 'g': __namedArgs.g, 'b': __namedArgs.b] as Color,
            ['type': __namedArgs.type, 'name': __namedArgs.name] as Animal)
}
```

@NamedParam work in progress

- Additional type checker support for @NamedParam
- Retrofitting @NamedParam onto existing methods

```
public static Sql newInstance(
    @NamedParam(value = 'url', type = String, required = true)
    @NamedParam(value = 'properties', type = Properties)
    @NamedParam(value = 'driverClassName', type = String)
    @NamedParam(value = 'driver', type = String)
    @NamedParam(value = 'user', type = String)
    @NamedParam(value = 'password', type = String)
    Map<String, Object> args
) throws SQLException, ClassNotFoundException {
    ...
}
```

@Newify enhanced with regex pattern

```
@Newify([Branch, Leaf])
def t = Branch(Leaf(1), Branch(Branch(Leaf(2), Leaf(3)), Leaf(4)))
assert t.toString() == 'Branch(Leaf(1), Branch(Branch(Leaf(2), Leaf(3)), Leaf(4)))'

@Newify(pattern='[BL].*')
def u = Branch(Leaf(1), Branch(Branch(Leaf(2), Leaf(3)), Leaf(5)))
assert u.toString() == 'Branch(Leaf(1), Branch(Branch(Leaf(2), Leaf(3)), Leaf(4)))'
```

Macros

- macro method, MacroClass
- AST matcher
- Macro methods (custom macros)

Without Macros

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

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

```
def ast = macro {
    return new Date()
}
```

With Macros (Groovy 2.5+)

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

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

```
def ast = macro {
    return new Date()
}
```

Macros (Groovy 2.5+)

- Variations:
  - Expressions, Statements, Classes
  - Supports variable substitution, specifying compilation phase

```
def varX = new VariableExpression('x')
def varY = new VariableExpression('y')

def pythagoras = macro {
  return Math.sqrt($v{varX} ** 2 + $v{varY} ** 2).intValue()
}
```

```
def pythagoras = macro(CANONICALIZATION, true) {
    Math.sqrt($v{varX} ** 2 + $v{varY} ** 2).intValue()
}
```

Macros (Groovy 2.5+)

- Variations:
  - Expressions, Statements, Classes
  - Supports variable substitution, specifying compilation phase

```groovy
@Statistics
class Person {
    Integer age
    String name
}

def p = new Person(age: 12,
                   name: 'john')

assert p.methodCount == 0
assert p.fieldCount  == 2
```

```groovy
ClassNode buildTemplateClass(ClassNode reference) {
    def methodCount = constX(reference.methods.size())
    def fieldCount = constX(reference.fields.size())

    return new MacroClass() {
        class Statistics {
            java.lang.Integer getMethodCount() {
                return $v { methodCount }
            }

            java.lang.Integer getFieldCount() {
                return $v { fieldCount }
            }
        }
    }
}
```

AST Matching

❖ AST Matching:
- Selective transformations, filtering, testing
- Supports placeholders

```
Expression transform(Expression exp) {
    Expression ref = macro { 1 + 1 }

    if (ASTMatcher.matches(ref, exp)) {
        return macro { 2 }
    }

    return super.transform(exp)
}
```

Macro method examples: match

```
def fact(num) {
    return match(num) {
        when String then fact(num.toInteger())
        when(0 | 1) then 1
        when 2 then 2
        orElse num * fact(num - 1)
    }
}

assert fact("5") == 120
```

See: https://github.com/touchez-du-bois/akatsuki

Macro method examples: doWithData

Spock inspired

```groovy
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

class MathSpec extends Specification {
    def "maximum of two numbers"(int a, int b, int c) {
        expect:
        Math.max(a, b) == c

        where:
        a | b | c
        1 | 3 | 3
        7 | 4 | 7
        0 | 0 | 0
    }
}
```

Macro method examples: doWithData

```
doWithData {
    dowith:
        assert a + b == c
    where:
        a | b || c
        1 | 2 || 3
        4 | 5 || 9
        7 | 8 || 15
}
```

See: https://github.com/touchez-du-bois/akatsuki

Misc features

- Repeated annotations
- Method parameter names
- Annotations in more places (JSR-308)
- :grab in groovysh
- tap in addition to with
- CliBuilder improvements
- JAXB marshalling shortcuts
- Customizable JSON serializer
- JUnit 5 runner support out of the box

:grab in groovysh

```
Groovy Shell (3.0.0-SNAPSHOT, JVM: 1.8.0_161)
Type ':help' or ':h' for help.
-------------------------------------------------
groovy:000> :grab 'com.google.guava:guava:24.1-jre'
groovy:000> import com.google.common.collect.ImmutableBiMap
===> com.google.common.collect.ImmutableBiMap
groovy:000> m = ImmutableBiMap.of('foo', 'bar')
===> [foo:bar]
groovy:000> m.inverse()
===> [bar:foo]
groovy:000>
```

With vs Tap

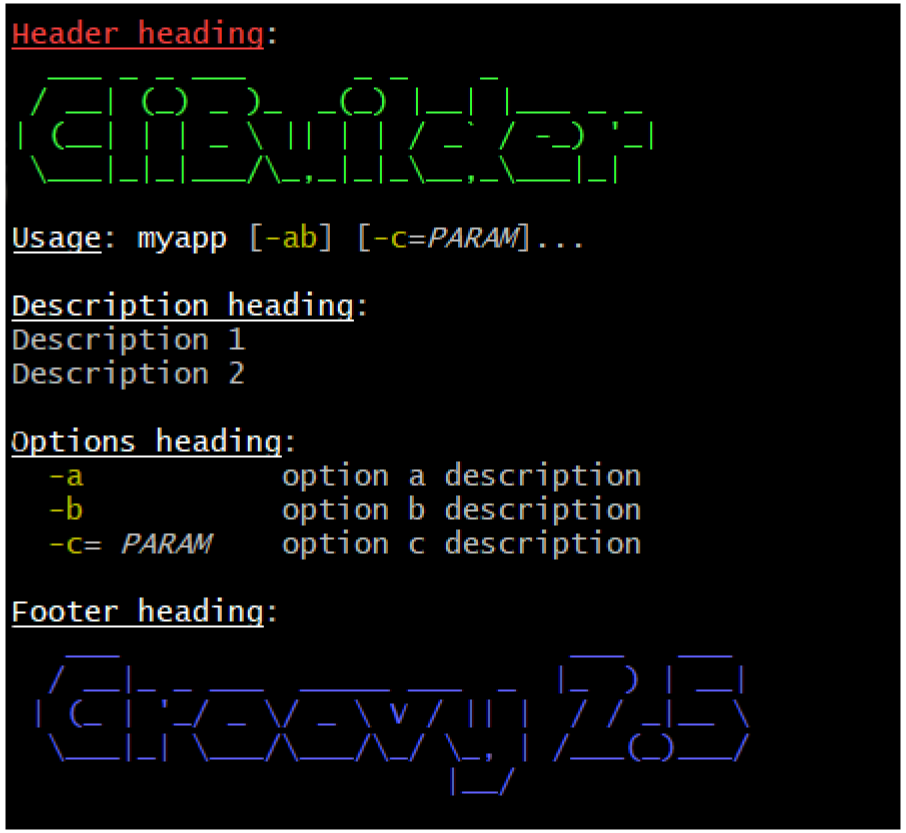
```
class Person {
    String first, last, honorific
    boolean friend
}

def p = new Person(last: 'Gaga', honorific: 'Lady', friend: false)
def greeting = 'Dear ' + p.with{ friend ? first : "$honorific $last" }
assert greeting == 'Dear Lady Gaga'

new Person().tap {
    friend = true
    first = 'Bob'
}.tap {
    assert friend && first || !friend && last
}.tap {
    if (friend) {
        println "Dear $first"
    } else {
        println "Dear $honorific $last"
    }
}
```

With vs Tap

```groovy
class Person {
    String first, last, honorific
    boolean friend
}

def p = new Person(last: 'Gaga', honorific: 'Lady', friend: false)
def greeting = 'Dear ' + p.with{ friend ? first : "$honorific $last" }
assert greeting == 'Dear Lady Gaga'

new Person().tap {
    friend = true
    first = 'Bob'
}.tap {
    assert friend && first || !friend && last
}.tap {
    if (friend) {
        println "Dear $first"
    } else {
        println "Dear $honorific $last"
    }
}
```

CliBuilder improvements

- ❖ Annotation support
- ❖ commons cli and picocli
- ❖ Improved typed options
- ❖ Improved converters
- ❖ Typed positional parameters
- ❖ Strongly typed maps
- ❖ Usage Help with ANSI Colors
- ❖ Tab autocompletion on Linux



```
Header heading:
  ___ _ _ ___      _ _    _
 / __| (_) _ )_  _(_) |__| |___ _ _
| (__| | | _ \ || | | / _` / -_) '_|
 \___|_|_|___/\_,_|_|_\__,_\___|_|

Usage: myapp [-ab] [-c=PARAM]...

Description heading:
Description 1
Description 2

Options heading:
  -a            option a description
  -b            option b description
  -c= PARAM     option c description

Footer heading:
  ___                        ___   ___
 / __|_ _ ___  _____ ___  _ |_  ) | __|
| (_ | '_/ _ \/ _ \ V / || | / / _|__ \
 \___|_| \___/\___/\_/ \_, |/___(_)___/
                       |__/
```

CliBuilder supports annotations

```groovy
interface GreeterI {
    @Option(shortName='h', description='display usage’)
    Boolean help()
    @Option(shortName='a', description='greeting audience’)
    String audience()
    @Unparsed
    List remaining()
}
```

```groovy
def cli = new CliBuilder(usage: 'groovy Greeter [option]')
def argz = '--audience Groovologist'.split()
def options = cli.parseFromSpec(GreeterI, argz)
assert options.audience() == 'Groovologist'
```

```groovy
@OptionField String audience
@OptionField Boolean help
@UnparsedField List remaining
new CliBuilder().parseFromInstance(this, args)
assert audience == 'Groovologist'
```

Junit 5 support via groovy and groovyConsole

```
class MyTest {
    @Test
    void streamSum() {
        assert Stream.of(1, 2, 3).mapToInt{ i -> i }.sum() > 5
    }

    @RepeatedTest(value=2, name = "{displayName} {currentRepetition}/{totalRepetitions}")
    void streamSumRepeated() {
        assert Stream.of(1, 2, 3).mapToInt{i -> i}.sum() == 6
    }

    private boolean isPalindrome(s) { s == s.reverse()  }

    @ParameterizedTest // requires org.junit.jupiter:junit-jupiter-params
    @ValueSource(strings = [ "racecar", "radar", "able was I ere I saw elba" ])
    void palindromes(String candidate) {
        assert isPalindrome(candidate)
    }

    @TestFactory
    def dynamicTestCollection() {[
            dynamicTest("Add test") { -> assert 1 + 1 == 2 },
            dynamicTest("Multiply Test") { -> assert 2 * 3 == 6 }
    ]}
}
```

```
JUnit5 launcher: passed=8, failed=0, skipped=0, time=246ms
```

Extensibility

GDK, runtime metaprogramming, operator overloading, *extensible type checker*, *compile-time metaprogramming*, *macros*:

- Let the Groovy team add bells and whistles to the language
- Allow you to do the same

Groovy 3.0 Themes

- **Parrot parser**
  - **Improved copy/paste with Java**
  - **New syntax/operators**
- Indy by default
- JDK8 minimum and better JDK 9+ JPMS support
- Additional documentation options

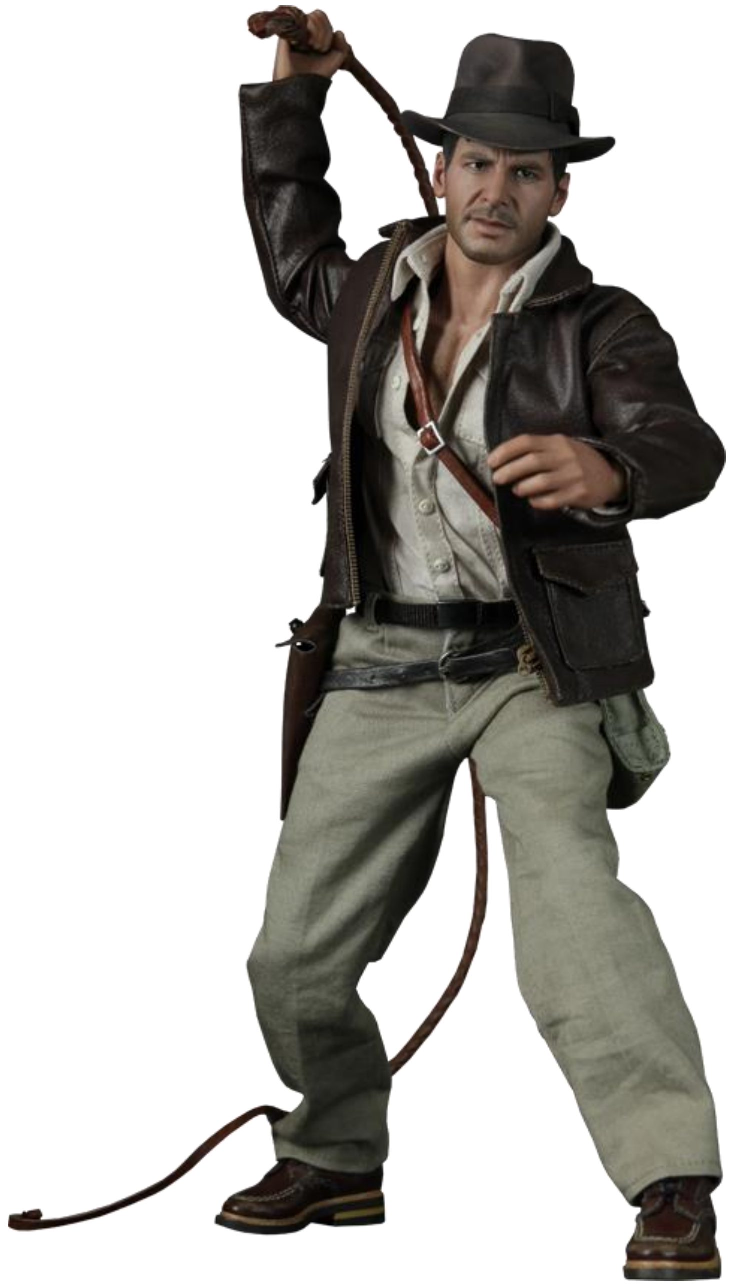
Groovy 3.0 Themes

- **Parrot parser**
  - **Improved copy/paste with Java**
  - **New syntax/operators**
- Indy by default
- JDK8 minimum and better JDK 9+ JPMS support
- Additional documentation options

Java you will be assimilated: resistance is futile

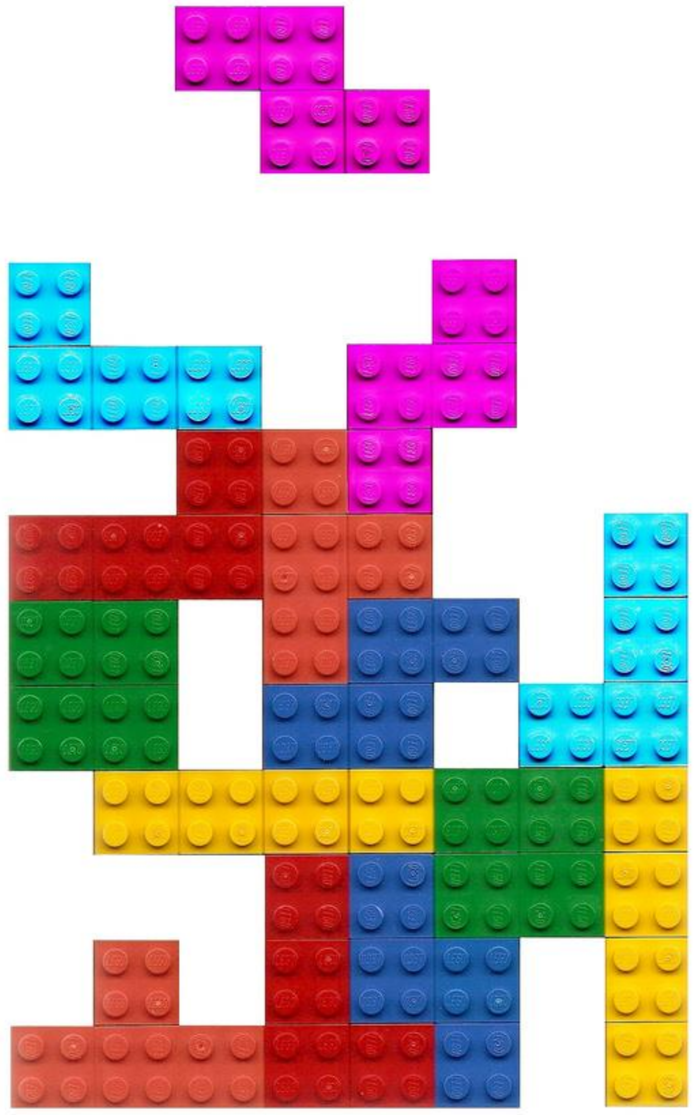
Groovy 3.0 Themes

- Parrot parser
  - Improved copy/paste with Java
  - New syntax/operators
- **Indy by default**
- JDK8 minimum and better JDK 9+ JPMS support
- Additional documentation options



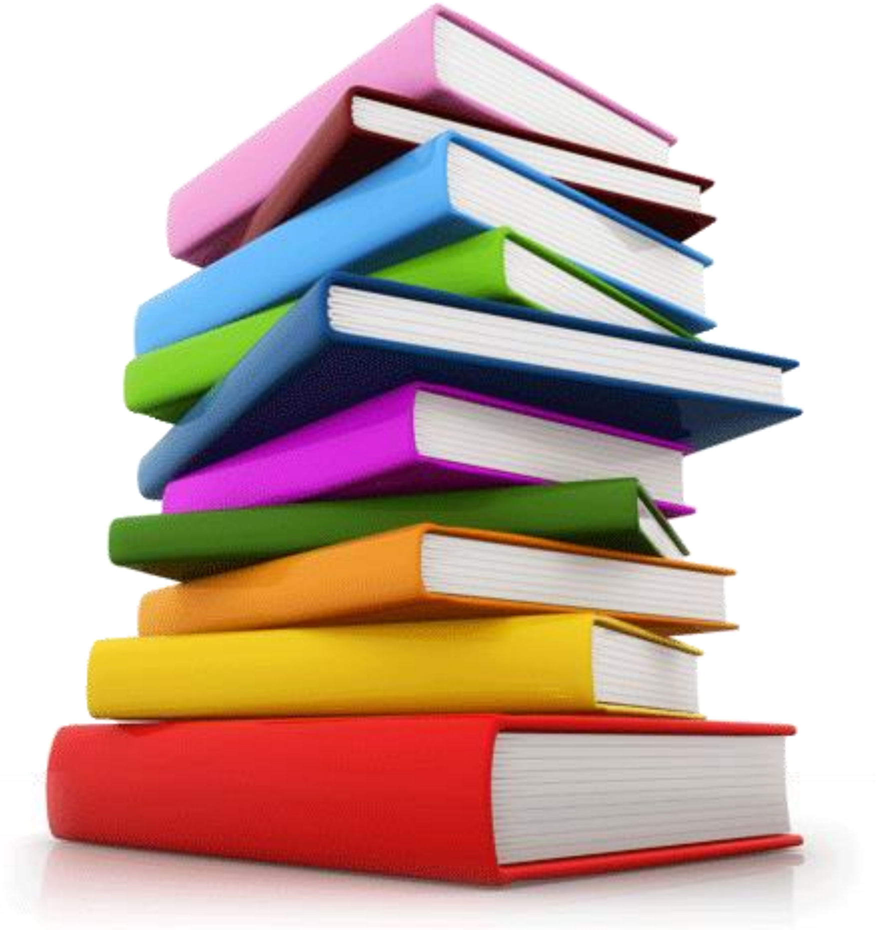
Groovy 3.0 Themes

- Parrot parser
  - Improved copy/paste with Java
  - New syntax/operators
- Indy by default
- **JDK8 minimum and better JDK 9+ JPMS support**
- Additional documentation options



Groovy 3.0 Themes

- Parrot parser
  - Improved copy/paste with Java
  - New syntax/operators
- Indy by default
- JDK8 minimum and better JDK 9+ JPMS support
- **Additional documentation options**



Parrot looping

```
// classic Java-style do..while loop
def count = 5
def fact = 1
do {
    fact *= count--
} while(count > 1)
assert fact == 120
```

Parrot looping

```
// classic for loop but now with extra commas
def facts = []
def count = 5
for (int fact = 1, i = 1; i <= count; i++, fact *= i) {
    facts << fact
}
assert facts == [1, 2, 6, 24, 120]
```

Parrot looping

```
// multi-assignment
def (String x, int y) = ['foo', 42]
assert "$x $y" == 'foo 42'

// multi-assignment goes loopy
def baNums = []
for (def (String u, int v) = ['bar', 42]; v < 45; u++, v++) {
    baNums << "$u $v"
}
assert baNums == ['bar 42', 'bas 43', 'bat 44']
```

Java-style array initialization

```
def primes = new int[] {2, 3, 5, 7, 11}
assert primes.size() == 5 && primes.sum() == 28
assert primes.class.name == '[I'

def pets = new String[] {'cat', 'dog'}
assert pets.size() == 2 && pets.sum() == 'catdog'
assert pets.class.name == '[Ljava.lang.String;'

// traditional Groovy alternative still supported
String[] groovyBooks = [ 'Groovy in Action', 'Making Java Groovy' ]
assert groovyBooks.every{ it.contains('Groovy') }
```

New operators: identity

```
import groovy.transform.EqualsAndHashCode

@EqualsAndHashCode
class Creature { String type }

def cat = new Creature(type: 'cat')
def copyCat = cat
def lion = new Creature(type: 'cat')

assert cat.equals(lion) // Java logical equality
assert cat == lion      // Groovy shorthand operator

assert cat.is(copyCat)  // Groovy identity
assert cat === copyCat  // operator shorthand
assert cat !== lion     // negated operator shorthand
```

New operators: negated variants

```
assert 45 !instanceof Date

assert 4 !in [1, 3, 5, 7]
```

New operators: Elvis assignment

```
import groovy.transform.ToString

@ToString
class Element {
    String name
    int atomicNumber
}
def he = new Element(name: 'Helium')
he.with {
    name = name ?: 'Hydrogen'   // existing Elvis operator
    atomicNumber ?= 2           // new Elvis assignment shorthand
}
assert he.toString() == 'Element(Helium, 2)'
```

Safe indexing

```
String[] array = ['a', 'b']
assert 'b' == array?[1]      // get using normal array index
array?[1] = 'c'              // set using normal array index
assert 'c' == array?[1]

array = null
assert null == array?[1]     // return null for all index values
array?[1] = 'c'              // quietly ignore attempt to set value
assert array == null
```

Better Java syntax support: try with resources

```
class FromResource extends ByteArrayInputStream {
    @Override
    void close() throws IOException {
        super.close()
        println "FromResource closing"
    }

    FromResource(String input) {
        super(input.toLowerCase().bytes)
    }
}

class ToResource extends ByteArrayOutputStream {
    @Override
    void close() throws IOException {
        super.close()
        println "ToResource closing"
    }
}
```

Better Java syntax support: try with resources

```
def wrestle(s) {
    try (
            FromResource from = new FromResource(s)
            ToResource to = new ToResource()
    ) {
        to << from
        return to.toString()
    }
}

assert wrestle("ARM was here!").contains('arm')
```

```
ToResource closing
FromResource closing
```

Better Java syntax support: nested blocks

```
{
    def a = 1
    a++
    assert 2 == a
}
try {
    a++ // not defined at this point
} catch(MissingPropertyException ex) {
    println ex.message
}
{
    {
        // inner nesting is another scope
        def a = 'banana'
        assert a.size() == 6
    }
    def a = 1
    assert a == 1
}
```

Better Java syntax support: var (JDK10/11)

- Local variables (JDK10)
- Lambda params (JDK11)

Lambdas

```
import static java.util.stream.Collectors.toList

(1..10).forEach(e -> { println e })

assert (1..10).stream()
        .filter(e -> e % 2 == 0)
        .map(e -> e * 2)
        .collect(toList()) == [4, 8, 12, 16, 20]
```

Lambdas – all the shapes

```
// general form
def add = (int x, int y) -> { def z = y; return x + z }
assert add(3, 4) == 7

// curly braces are optional for a single expression
def sub = (int x, int y) -> x - y
assert sub(4, 3) == 1

// parameter types and
// explicit return are optional
def mult = (x, y) -> { x * y }
assert mult(3, 4) == 12
```

```
// no parentheses required for a single parameter with no type
def isEven = n -> n % 2 == 0
assert isEven(6)
assert !isEven(7)

// no arguments case
def theAnswer = () -> 42
assert theAnswer() == 42

// any statement requires braces
def checkMath = () -> { assert 1 + 1 == 2 }
checkMath()

// example showing default parameter values (no Java equivalent)
def addWithDefault = (int x, int y = 100) -> x + y
assert addWithDefault(1, 200) == 201
assert addWithDefault(1) == 101
```

Method references: instances

```
// instance::instanceMethod
def sizeAlphabet =
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'::length
assert sizeAlphabet() == 26

// instance::staticMethod
def hexer = 42::toHexString
assert hexer(127) == '7f'
```

Currently implemented
as method closures

Method references: classes

```
import java.util.stream.Stream
import static java.util.stream.Collectors.toList

// class::staticMethod
assert ['1', '2', '3'] ==
        Stream.of(1, 2, 3)
                .map(String::valueOf)
                .collect(toList())

// class::instanceMethod
assert ['A', 'B', 'C'] ==
        ['a', 'b', 'c'].stream()
                .map(String::toUpperCase)
                .collect(toList())
```

Method references: constructors

```groovy
// normal constructor
def r = Random::new
assert r().nextInt(10) in 0..9

// array constructor is handy when working with various Java libraries, e.g. streams
assert [1, 2, 3].stream().toArray().class.name == '[Ljava.lang.Object;'
assert [1, 2, 3].stream().toArray(Integer[]::new).class.name == '[Ljava.lang.Integer;'

// works with multi-dimensional arrays too
def make2d = String[][]::new
def tictac = make2d(3, 3)
tictac[0] = ['X', 'O', 'X']
tictac[1] = ['X', 'X', 'O']
tictac[2] = ['O', 'X', 'O']
assert tictac*.join().join('\n') == '''
XOX
XXO
OXO
'''.trim()
```

Method references: constructors

```
// also useful for your own classes
import groovy.transform.Canonical
import java.util.stream.Collectors

@Canonical
class Animal {
    String kind
}

def a = Animal::new
assert a('lion').kind == 'lion'

def c = Animal
assert c::new('cat').kind == 'cat'

def pets = ['cat', 'dog'].stream().map(Animal::new)
def names = pets.map(Animal::toString).collect(Collectors.joining( "," ))
assert names == 'Animal(cat),Animal(dog)'
```

Default methods in interfaces

WORK IN PROGRESS

```
interface Greetable {
    String target()

    default String salutation() {
        'Greetings'
    }

    default String greet() {
        "${salutation()}, ${target()}"
    }
}

class Greetee implements Greetable {
    String name
    @Override
    String target() { name }
}

def daniel = new Greetee(name: 'Daniel')
assert 'Greetings, Daniel' == "${daniel.salutation()}, ${daniel.target()}"
assert 'Greetings, Daniel' == daniel.greet()
```

Currently implemented using traits

GroovyDoc comments as metadata

```
import org.codehaus.groovy.control.*

import static groovy.lang.groovydoc.GroovydocHolder.DOC_COMMENT

def ast = new CompilationUnit().tap {
    addSource 'myScript.groovy', '''
        /** class doco */
        class MyClass {
            /** method doco */
            def myMethod() {}
        }
    '''
    compile Phases.SEMANTIC_ANALYSIS
}.ast

def classDoc = ast.classes[0].groovydoc
assert classDoc.content.contains('class doco')
def methodDoc = ast.classes[0].methods[0].groovydoc
assert methodDoc.content.contains('method doco')
```

Requires: -Dgroovy.attach.groovydoc=true

Groovydoc comments: runtime embedding

```
class Foo {
    /** @Groovydoc fo fum */
    def bar() { }
    @Groovydoc('Hard-coded')
    def baz() { }
}

def docForMethod(String name) {
//    Foo.methods.find{ it.name == name }.getAnnotation(Groovydoc).value()
    Foo.methods.find{ it.name == name }.groovydoc.content
}

assert docForMethod('bar').contains('@Groovydoc fo fum')
assert docForMethod('baz').contains('Hard-coded')
```

Finish with a Monty Python inspired summary

… but apart from the sanitation, the medicine, education, wine, public order, irrigation, roads, a fresh water system, and public health, **what have the Romans ever done for us**?

Groovy is just some syntactic sugar over Java

Some of the Groovy team circa 2006:
Guillaume Laforge, Jochen Theodorou,
Dierk Koenig, Jeremy Rayner,
John Wilson and James Strachan

… apart from the GDK, operator overloading, ranges, runtime & compile-time metaprogramming, elvis, extensible type checker, Closures, macros, traits, ranges, scripts, native regex Operators, GPath, builders, command chains, named params, …

Join us: groovy.apache.org