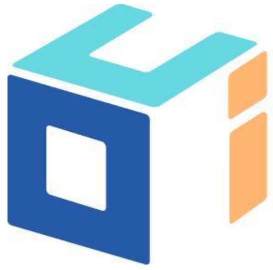


Groovy 2.5 features & 3+ Roadmap

Dr Paul King

OCI Groovy Lead / V.P. and PMC Chair Apache Groovy

https://speakerdeck.com/paulk/groovy-roadmap

https://github.com/paulk-asert/upcoming-groovy

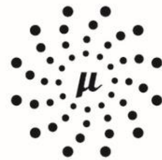
@paulk_asert

© 2018, Object Computing, Inc. (OCI). All rights reserved. No part of these notes may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior, written permission of Object Computing, Inc. (OCI)

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.



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

© 2018, Object Computing, Inc. (OCI). All rights reserved.

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.

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

- Architecture consulting
- Blockchain consulting
- Cloud solutions
- Cyber security
- Devops solutions
- Groovy & Grails
- Gradle consulting
- Internet of Things (IoT)
- Kubernetes consulting
- Machine Learning
- Micronaut and Microservices solutions
- Mobile development
- Real-time and embedded systems development
- Software Engineering
- Technology Training
- Web Development

© 2018, Object Computing, Inc. (OCI). All rights reserved.

Why Groovy?

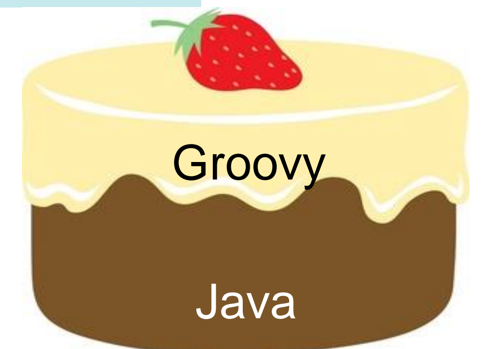
Some common languages before Groovy was born

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

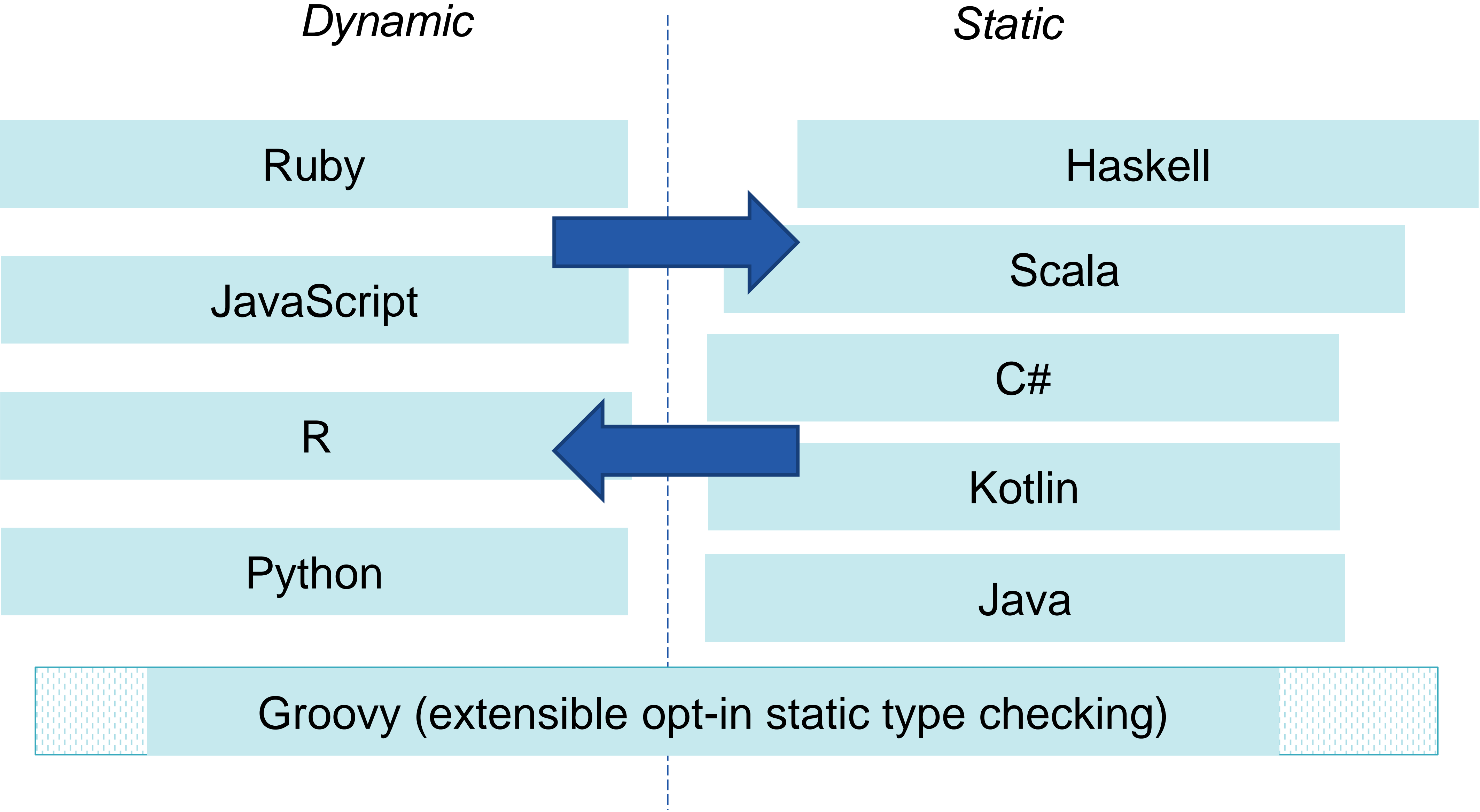
Some common languages when Groovy was born

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

Groovy



Now a spectrum



What is Groovy?

Groovy is like a super version of Java.

It leverages existing Java capabilities but adds productivity features and provides great flexibility and extensibility.

Groovy as a Java superset
It’s so easy to learn!
Groovy
Groovy
Groovy

Java code for list manipulation

```
import java.util.List;
import java.util.ArrayList;

class Main {
    private List keepShorterThan(List strings, int length) {
        List result = new ArrayList();
        for (int i = 0; i < strings.size(); i++) {
            String s = (String) strings.get(i);
            if (s.length() < length) {
                result.add(s);
            }
        }
        return result;
    }
    public static void main(String[] args) {
        List names = new ArrayList();
        names.add("Ted"); names.add("Fred");
        names.add("Jed"); names.add("Ned");
        System.out.println(names);
        Main m = new Main();
        List shortNames = m.keepShorterThan(names, 4);
        System.out.println(shortNames.size());
        for (int i = 0; i < shortNames.size(); i++) {
            String s = (String) shortNames.get(i);
            System.out.println(s);
        }
    }
}
```

Groovy code for list manipulation

```
import java.util.List;
import java.util.ArrayList;

class Main {
    private List keepShorterThan(List strings, int length) {
        List result = new ArrayList();
        for (int i = 0; i < strings.size(); i++) {
            String s = (String) strings.get(i);
            if (s.length() < length) {
                result.add(s);
            }
        }
        return result;
    }
    public static void main(String[] args) {
        List names = new ArrayList();
        names.add("Ted"); names.add("Fred");
        names.add("Jed"); names.add("Ned");
        System.out.println(names);
        Main m = new Main();
        List shortNames = m.keepShorterThan(names, 4);
        System.out.println(shortNames.size());
        for (int i = 0; i < shortNames.size(); i++) {
            String s = (String) shortNames.get(i);
            System.out.println(s);
        }
    }
}
```

Rename `Main.java` *to* `Main.groovy`

Some Java Boilerplate identified

```
import java.util.List;
import java.util.ArrayList;

class Main {
    private List keepShorterThan(List strings, int length) {
        List result = new ArrayList();
        for (int i = 0; i < strings.size(); i++) {
            String s = (String) strings.get(i);
            if (s.length() < length) {
                result.add(s);
            }
        }
        return result;
    }
    public static void main(String[] args) {
        List names = new ArrayList();
        names.add("Ted"); names.add("Fred");
        names.add("Jed"); names.add("Ned");
        System.out.println(names);
        Main m = new Main();
        List shortNames = m.keepShorterThan(names, 4);
        System.out.println(shortNames.size());
        for (int i = 0; i < shortNames.size(); i++) {
            String s = (String) shortNames.get(i);
            System.out.println(s);
        }
    }
}
```

Are the semicolons needed?
And shouldn't we us more modern list notation?
Why not import common libraries?
Do we need the static types?
Must we always have a main method and class definition?
How about improved consistency?

Java Boilerplate removed

```
def keepShorterThan(strings, length) {
    def result = new ArrayList()
    for (s in strings) {
        if (s.size() < length) {
            result.add(s)
        }
    }
    return result
}

names = new ArrayList()
names.add("Ted"); names.add("Fred")
names.add("Jed"); names.add("Ned")
System.out.println(names)
shortNames = keepShorterThan(names, 4)
System.out.println(shortNames.size())
for (s in shortNames) {
    System.out.println(s)
}
```

More Java Boilerplate identified

```
def keepShorterThan(strings, length) {
    def result = new ArrayList()
    for (s in strings) {
        if (s.size() < length) {
            result.add(s)
        }
    }
    return result
}

names = new ArrayList()
names.add("Ted"); names.add("Fred")
names.add("Jed"); names.add("Ned")
System.out.println(names)
shortNames = keepShorterThan(names, 4)
System.out.println(shortNames.size())
for (s in shortNames) {
    System.out.println(s)
}
```

Shouldn't we have special notation for lists?

And special facilities for list processing?

Is 'return' needed at end?

Is the method now needed?

Simplify common methods?

Remove unambiguous brackets?

Boilerplate removed = nicer Groovy version

```
names = ["Ted", "Fred", "Jed", "Ned"]
println names
shortNames = names.findAll{ it.size() < 4 }
println shortNames.size()
shortNames.each{ println it }
```

Output:

```
["Ted", "Fred", "Jed", "Ned"]
3
Ted
Jed
Ned
```

Boilerplate removed = nicer Groovy version

```groovy
names = ["Ted", "Fred", "Jed", "Ned"]
println names
shortNames = names.findAll{ it.size() < 4 }
println shortNames.size()
shortNames.each{ println it }
```

```java
import java.util.List;
import java.util.stream.Collectors;

class Main {
    public static void main(String[] args) {
        var names = List.of("Ted", "Fred", "Jed", "Ned");
        System.out.println(names);
        var shortNames = names.stream()
                .filter(name -> name.length() < 4)
                .collect(Collectors.toList());
        System.out.println(shortNames.size());
        shortNames.forEach(System.out::println);
    }
}
```

Java 11 / Groovy 3.0

Or Groovy DSL version if required

```
given the names "Ted", "Fred", "Jed" and "Ned"
display all the names
display the number of names having size less than 4
display the names having size less than 4
```

```
// plus a DSL implementation
```

Or Groovy DSL version if required

```
given the names "Ted", "Fred", "Jed" and "Ned"
display all the names
display the number of names having size less than 4
display the names having size less than 4
```

```
names = []
def of, having, less
def given(_the) { [names:{ Object[] ns -> names.addAll(ns)
  [and: { n -> names += n }] }] }
def the = [
  number: { _of -> [names: { _having -> [size: { _less -> [than: { size ->
    println names.findAll{ it.size() < size }.size() }]}] }] },
  names: { _having -> [size: { _less -> [than: { size ->
    names.findAll{ it.size() < size }.each{ println it } }]}] }
]
def all = [the: { println it }]
def display(arg) { arg }
```

Or Groovy DSL version if required

```
given the names "Ted", "Fred", "Jed" and "Ned"
display all the names
display the number of names having size less than 4
display the names having size less than 4
```

```
display the names having size less than 4
```

Cannot resolve symbol 'names'

```
display the names having size less than 4
```

Add Dynamic Method 'names(Object)'

Or use GDSL (IntelliJ IDEA) or DSLD (Eclipse)

Or typed Groovy DSL version if required

```
given the names "Ted", "Fred", "Jed" and "Ned"
display all the names
display the number of names having size less than 4
display the names having size less than 4
```

```
…
enum The { the }
enum Having { having }
enum Of { of }
…
class DisplayThe {
    DisplayTheNamesHaving names(Having having) {
        new DisplayTheNamesHaving()
    }
    DisplayTheNumberOf number(Of of) {
        new DisplayTheNumberOf()
    }
}
…
// plus 50 lines
```

Or typed Groovy DSL version if required

```
given the names "Ted", "Fred", "Jed" and "Ned"
display all the names
display the number of names having size less than 4
display the names having size less than 4
```

The the
All all

```
display the names having size less than 4
names(Having having)          DisplayTheNamesHaving
number(Of of)                 DisplayTheNumberOf
```

```
given the names "Ted", "Fred", "Jed" and "Ned"
display name(String singleName)                       void
display names(String[] listOfAllNamesButLast)         AndConnector
```

Groovy DSL being debugged

```
74  class DisplayTheNumberOfNamesHavingSizeLess {
75      void than(int size) {
76          println MainScriptTypedDSL.names.findAll { it.size() < size }.size()
77      }
78  }
79
80  given the names "Ted", "Fred", "Jed" and "Ned"
81  display all the names
82  display the number of names having size less than 4
83  display the names having size less than 4
84
```

Debug MainScriptTypedDSL

Debugger Console

Frames

"main"@1 in group "main": RUNNING

than():74, DisplayTheNumberOfNamesHavingSizeLess

Variables

this = {DisplayTheNumberOfNar

size = 4

Or typed Groovy DSL version if required

```
@TypeChecked(extensions='EdChecker.groovy')
def method() {
    given the names "Ted", "Fred", "Jed" and "Ned"
    display all the names
    display the number of names having size less than 4
    display the names having size less than 4
}
```

Or typed Groovy DSL version if required

```
@TypeChecked(extensions='EdChecker.groovy')
def method() {
    given the names "Ted", "Fred", "Jed" and "Ned"
    display all the names
    display the number of names having size less than 4
    display the names having size less than 4
}
```

```
afterMethodCall { mc ->
  mc.arguments.each {
    if (isConstantExpression(it)) {
      if (it.value instanceof String && !it.value.endsWith('ed')) {
        addStaticTypeError("I don't like the name '${it.value}'", mc)
      }
    }
  }
}
```

Or typed Groovy DSL version if required

```
@TypeChecked(extensions='EdChecker.groovy')
def method() {
    given the names "Ted", "Fred", "Jed" and "Ned"
    display all the names
    display the number of names having size less than 4
    display the names having size less than 4
}
```

```
afterMethodCall { mc ->
  mc.arguments.each {
    if (isConstantExpression(it)) {
      if (it.value instanceof String && !it.value.endsWith('ed')) {
        addStaticTypeError("I don't like the name '${it.value}'", mc)
      }
    }
  }
}
```

Or typed Groovy DSL version if required

```
@TypeChecked(extensions='EdChecker.groovy')
def method() {
    given the names "Ted", “Mary", "Jed" and “Pete"
    display all the names
    display the number of names having size less than 4
    display the names having size less than 4
}
```

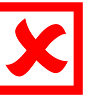
```
afterMethodCall { mc ->
  mc.arguments.each {
    if (isConstantExpression(it)) {
      if (it.value instanceof String && !it.value.endsWith('ed')) {
        addStaticTypeError("I don't like the name '${it.value}'", mc)
      }
    }
  }
}
```

Or typed Groovy DSL version if required

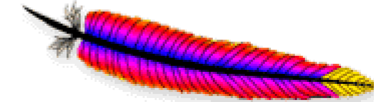
```
@TypeChecked(extensions='EdChecker.groovy')
def method() {
    given the names "Ted", “Mary", "Jed" and “Pete"
    display all the names
    display the number of names having size less than 4
    display the names having size less than 4
}
```

```
afterMethodCall { mc ->
  mc.arguments.each {
    if (isConstantExpression(it)) {
      if (it.value instanceof String && !it.value.endsWith('ed')) {
        addStaticTypeError("I don't like the name '${it.value}'", mc)
      }
    }
  }
}
```

```
[Static type checking] - I don't like the name 'Mary'
 at line: 83, column: 21

[Static type checking] - I don't like the name 'Pete'
 at line: 83, column: 5
```



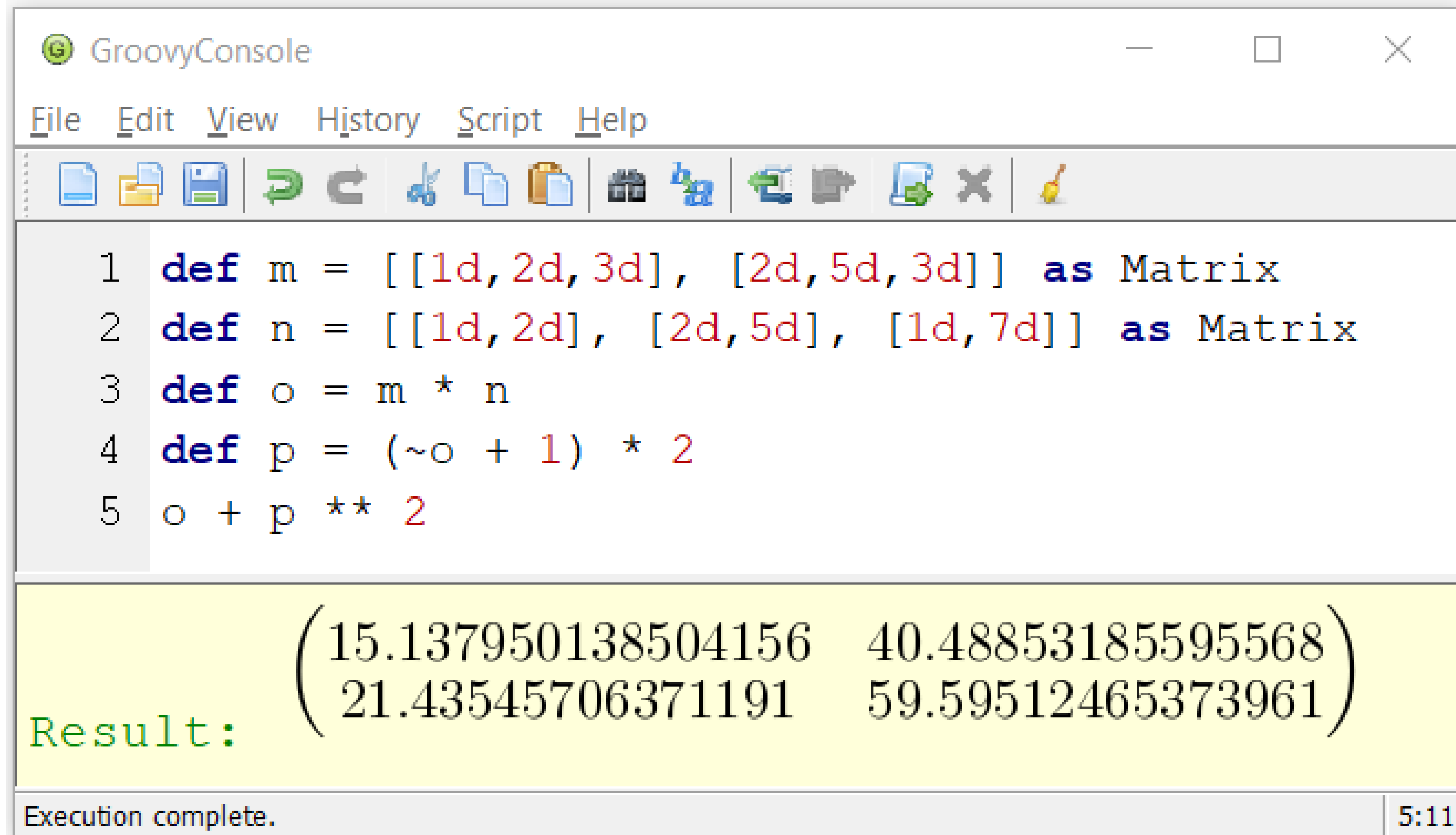
Matrix manipulation example




```
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);
    }
}
```

Array2DRowRealMatrix{{15.1379501385,40.488531856},{21.4354570637,59.5951246537}}

Operator overloading and extensible tools



What is Groovy?

```
Groovy = Java
   – boiler plate code
   + productivity enhancements
```

What is Groovy?

Groovy = Java
- boiler plate code
+ **extensible dynamic and static natures** (optional typing, extensible static typing)
+ **better functional programming** (closures)
+ **better OO programming** (properties, traits)

What is Groovy?

```
Groovy = Java
   – boiler plate code
   + extensible dynamic and static natures
   + better functional programming
   + better OO programming
   + runtime metaprogramming
     (metaclass, lifecycle methods, extensions methods)
   + compile-time metaprogramming
     (AST transforms, macros, extension methods)
   + operator overloading
```

What is Groovy?

```
Groovy = Java
   – boiler plate code
   + extensible dynamic and static natures
   + better functional programming
   + better OO programming
   + runtime metaprogramming
     (metaclass, lifecycle methods, extensions methods)
   + compile-time metaprogramming
     (AST transforms, macros, extension methods)
   + operator overloading
```

2.4 includes 49 transforms
2.5 includes 60 transforms
3.0 includes 61+ transforms

2.4 includes 1350 GDK methods
2.5 includes 1640 GDK methods
3.0 includes 1670+ GDK methods

What is Groovy?

Groovy = Java
- – boiler plate code
- \+ extensible dynamic and static natures
- \+ better functional programming
- \+ better OO programming
- \+ runtime metaprogramming
- \+ compile-time metaprogramming
- \+ operator overloading
- **+ additional tools** (groovydoc, groovyConsole, groovysh)
- **+ additional productivity libraries** (Regex, XML, SQL, JSON, YAML, CLI, builders)
- **+ scripts & flexible language grammar** (DSLs)

Groovy by the Numbers

- 2.4 in maintenance, 2.5 current, 3.0 in development
- Popular and growing
  2016: 23M
  2017: 50M
  currently: approx. 9M+ per month
- 18 releases and 40+ new contributors in last 12 months
- Could do with even more contributors! ☺

Groovy Roadmap

❖ Groovy 2.5

- 2.5.4 released, 2.5.5 soon
- Macros, AST transformation improvements, various misc. features
- JDK 7 minimum, runs on JDK 9/10/11 with warnings

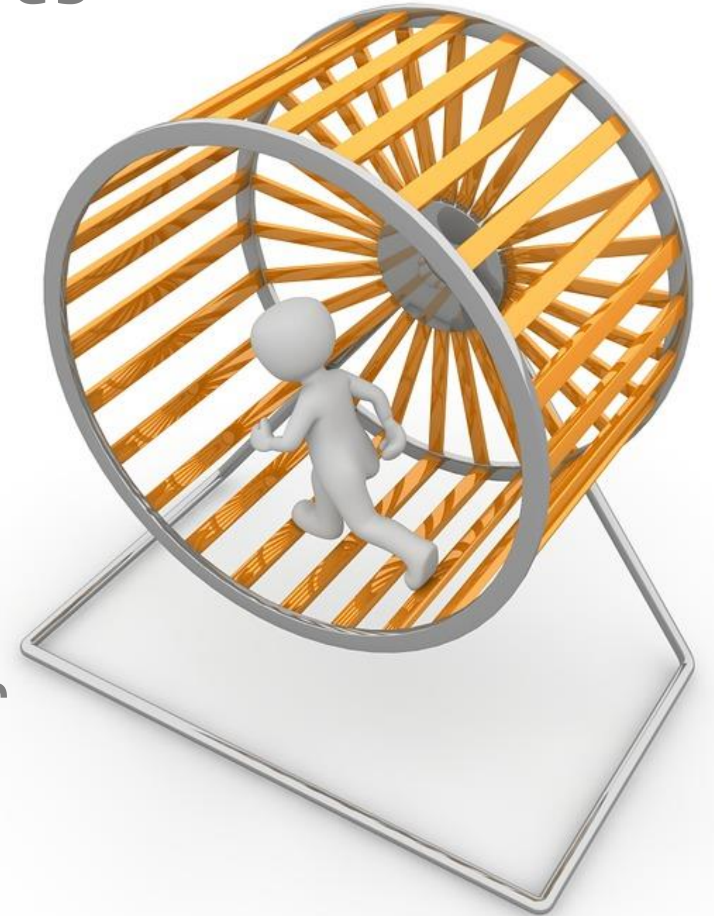
❖ Groovy 3.0

- 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 issues

What is Groovy?

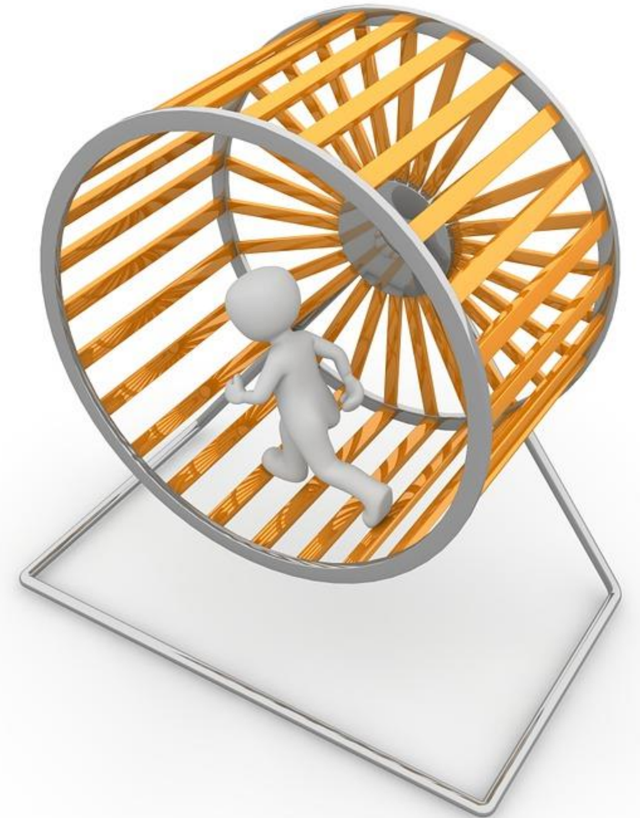
```
Groovy = Java
  – boiler pl te code
  + extensible     namic and static natures
  + better f                        ing
  + better O
  + runtime
  + compile-time metaprogramming
  + operator overloading
  + additional tools
  + additional productivity libraries
  + scripts & flexible language grammar
```

As Java evolves,
Groovy must evolve too!



Groovy 2.5 Java compatibility enhancements

- Repeated annotations (JEP 120 JDK8)
- Method parameter names (JEP 118 JDK8)
- Annotations in more places (JSR 308)
- Runs on JDK 9/10/11 (currently with warnings)
- Repackaging towards JPMS



Repeated annotations (JEP 120)

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Repeatable(MyAnnotationArray)
@interface MyAnnotation {
    String value() default "val0"
}
```

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnnotationArray {
    MyAnnotation[] value()
}
```

```
@MyAnnotationArray([
    @MyAnnotation("val1"),
    @MyAnnotation("val2")])
String method1() { 'method1' }
```

```
@MyAnnotation("val1")
@MyAnnotation("val2")
String method2() { 'method2' }
```

Method Parameter names (JEP 118)

```
class Foo {
    def foo(String one, Integer two, Date three) {}
}
```

$ groovyc Foo.groovy
$ groovy -e "println Foo.methods.find{ it.name == ‘foo' }.parameters"
[java.lang.String **arg0**, java.lang.Integer **arg1**, java.util.Date **arg2**]

```
class Bar {
    def bar(String one, Integer two, Date three) {}
}
```

$ groovyc **-parameters** Bar.groovy
$ groovy -e "println Bar.methods.find{ it.name == 'bar' }.parameters"
[java.lang.String **one**, java.lang.Integer **two**, java.util.Date **three**]

Work in progress…annotations in more places (JSR 308)

```
class MyException extends @Critical Exception {
    ...
}

@Readonly Map<F extends @Existing File, @NonNegative Integer> sizes

String @NonNull [] @NonEmpty [] @ReadOnly [] items
```

Partial support in 2.5, further support in Parrot parser

Modules – Groovy 2.4, Groovy 2.5, Groovy 3.0

groovy
groovy-all *fat jar -> fat pom*
groovy-ant
groovy-bsf *optional*
groovy-cli-commons *optional*
groovy-cli-picocli
groovy-console
groovy-datetime *from groovy*
groovy-dateutil *optional*
groovy-docgenerator
groovy-groovydoc
groovy-groovysh
groovy-jaxb *optional*
groovy-jmx

groovy-json
groovy-json-direct *optional* *removed*
groovy-jsr223
groovy-macro
groovy-nio
groovy-servlet
groovy-sql
groovy-swing
groovy-templates
groovy-test
groovy-test-junit5
groovy-testng
groovy-xml
groovy-yaml *optional*

What is Groovy?

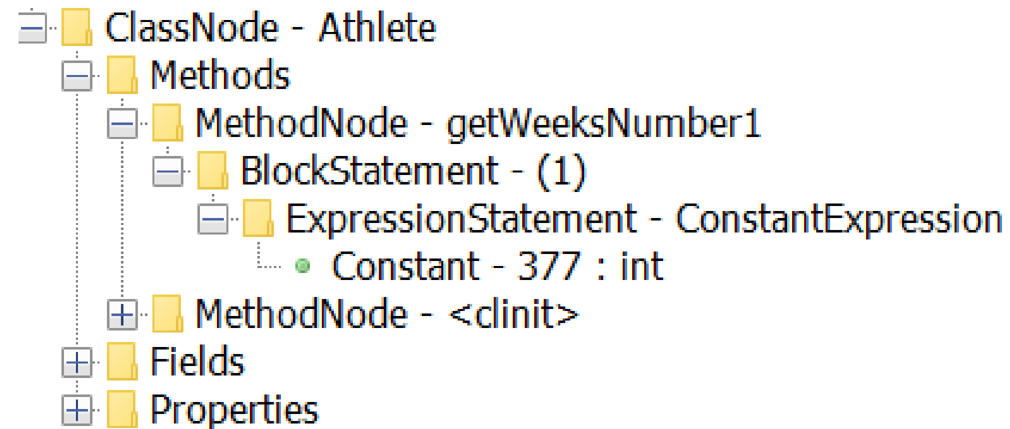
```
Groovy = Java
   – boiler plate code
   + extensible dynamic and static natures
   + better functional programming
   + better OO programming
   + runtime metaprogramming
   + compile-time metaprogramming
     (AST transforms, macros, extension methods)
   + operator overloading
   + additional         ls
   + additiona                    ries
   + scripts &                     rammar
```

Some additions and consistency improvements!

Groovy compilation process

- Multiple phases
- Skeletal AST

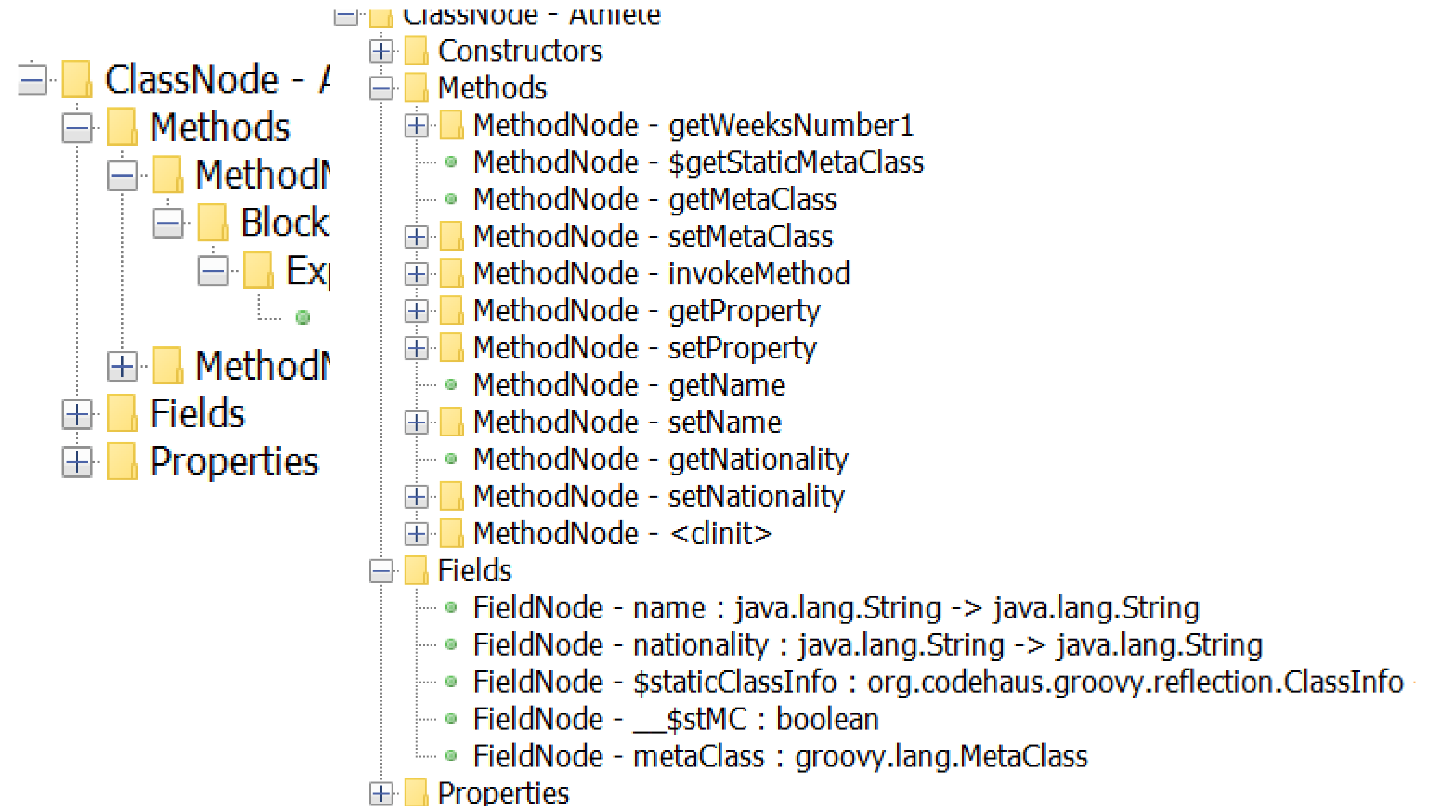
```
class Athlete {
    String name, nationality
    int getWeeksNumber1() {
        377
    }
}

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



Groovy compilation process

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

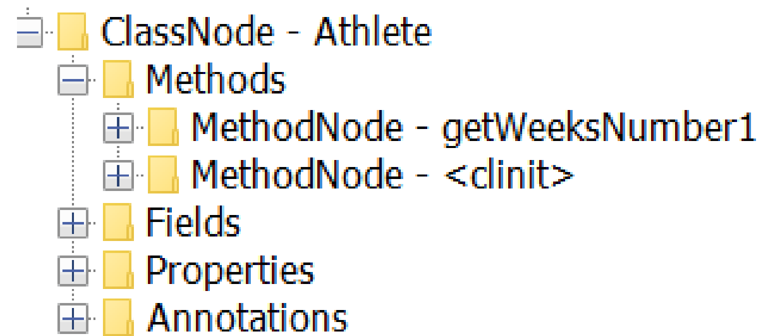
```
class Athlete {
    String name, nationality
    int getWeeksNumber1() {
        377
    }
}

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



Compile-time metaprogramming: AST transformations

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

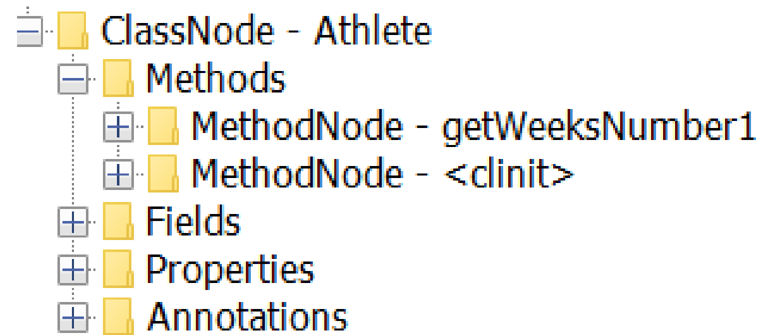
```
@ToString
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
}

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

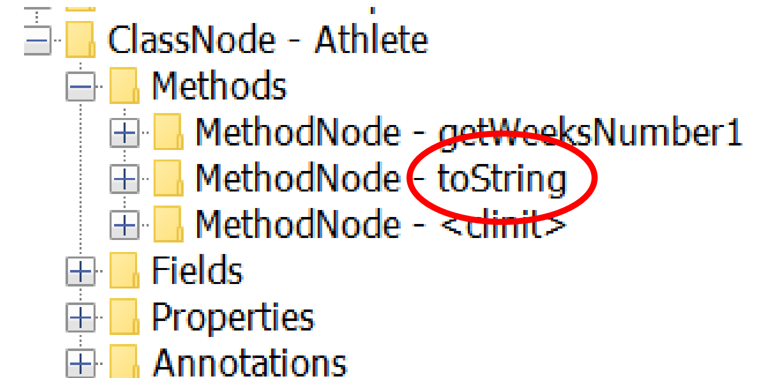


Compile-time metaprogramming: AST transformations

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

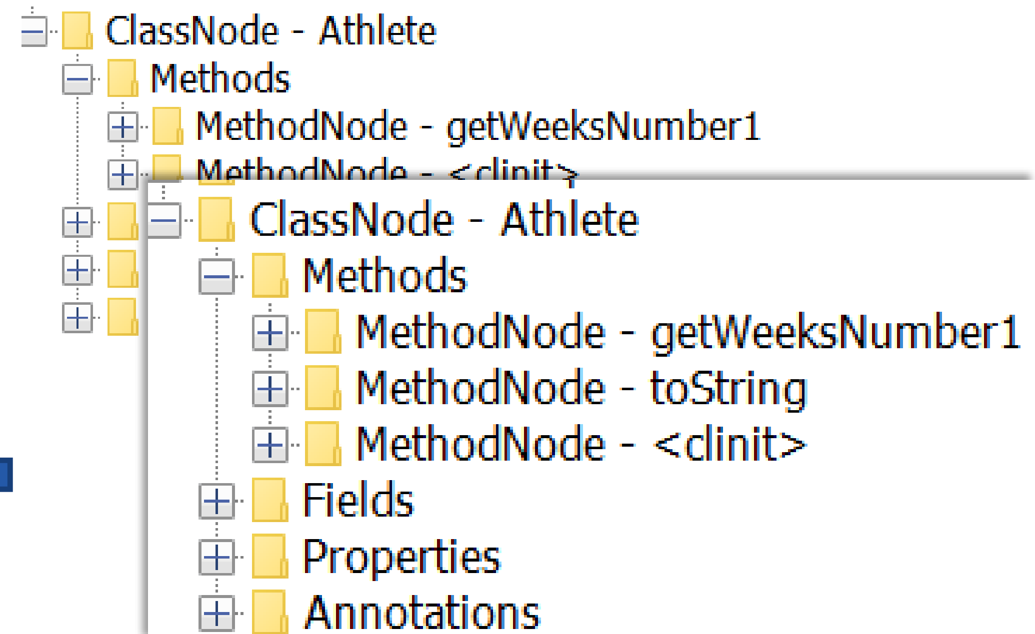
```
@ToString
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
}

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

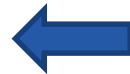




Compile-time metaprogramming: AST transformations

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

new At
      n
```



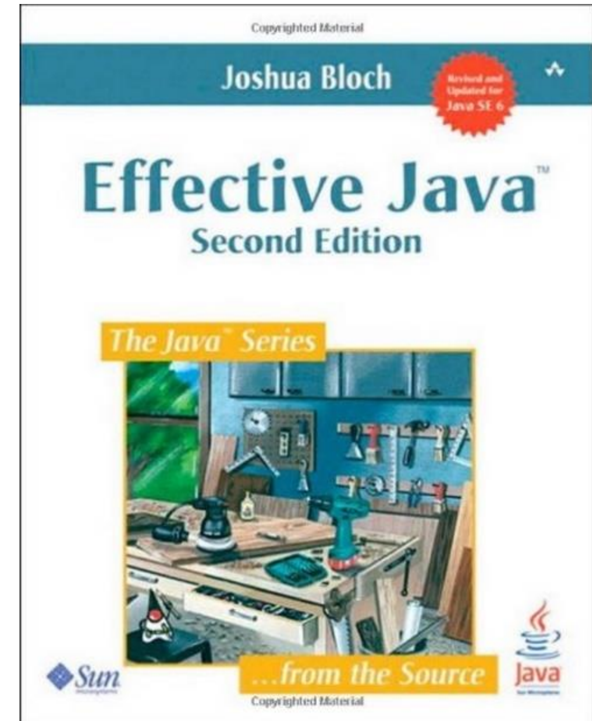


```
class Athlete {
    String name, nationality
    int getWeeksNumber1() { 377 }
    String toString() {
        def sb = new StringBuilder()
        sb << 'Athlete('
        sb << name
        sb << ', '
        sb << nationality
        sb << ')'
        return sb.toString()
    }
}
```

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…

Java Immutable Class

- As per Joshua Bloch Effective Java

boilerplate

```
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;
    }
    // ...
```

```
    // ...
    @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
```

@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

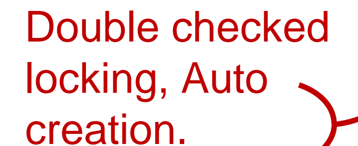
```
class Resource{}     // expensive resource
```

```
def res1 = new Resource()

@Lazy res2 = new Resource()

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

@Lazy volatile Resource res4

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

@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
```

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
}
```

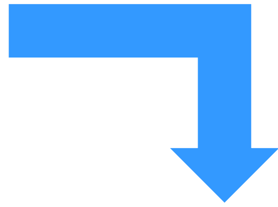
```
@ToString(cache = true, includeNames = true)
@TupleConstructor(useSetters = true)
@EqualsAndHashCode(cache = true)
class Point {
    int x, y
}
```

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

```
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())
```

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)
```

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
}
```

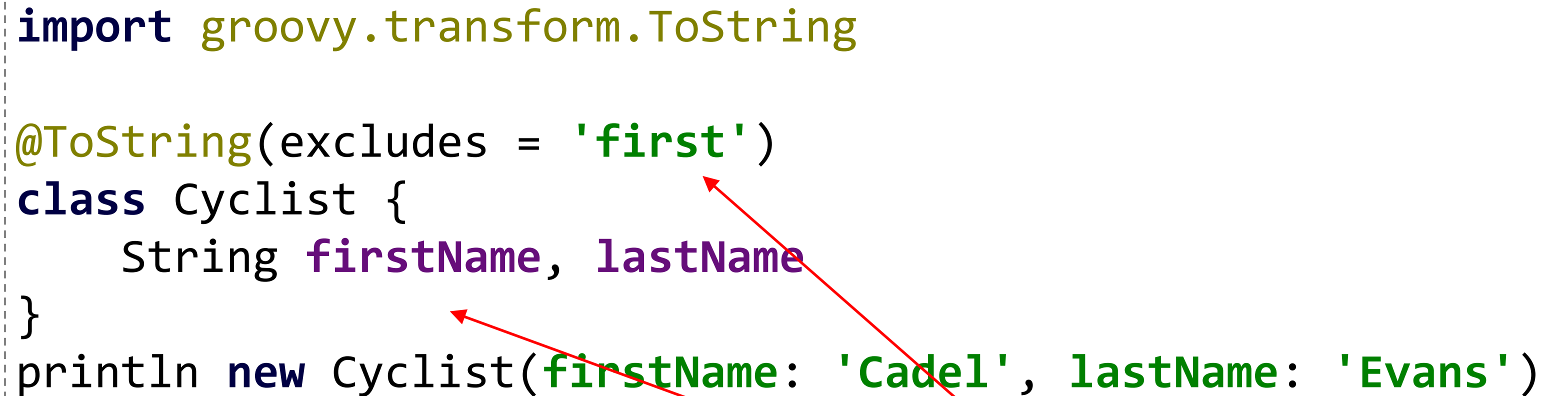
@Immutable processor doesn't know how to handle field 'origin' of type 'java.util.Optional' while compiling class Template…

AST Transforms: property name validation

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

```
import groovy.transform.ToString

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

AST Transforms: property name validation



- 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 defaults

```
import groovy.transform.TupleConstructor

//@TupleConstructor(defaults = false)
class Flight {
    Flight(String fromCity, String toCity, Date leaving){/* ... */}
    String fromCity, toCity
    Date leaving
}

//@TupleConstructor(defaults = true)
class Cruise {
    Cruise(String fromPort=null, String toPort=null, Date leaving=null){/* ... */}
    String fromPort, toPort
    Date leaving
}

println Flight.constructors.j
println Cruise.constructors.join('\n')
```

public Flight(java.lang.String,java.lang.String,java.util.Date)

public Cruise()
public Cruise(java.lang.String)
public Cruise(java.lang.String,java.lang.String)
public Cruise(java.lang.String,java.lang.String,java.util.Date)

AST Transforms: @TupleConstructor pre/post

```
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

```
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
}
```

@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)))'
```

What is Groovy?

Groovy = Java
- – boiler plate code
- \+ extensible dynamic and static natures
- \+ better functional programming
- \+ better OO programming
- \+ runtime metaprogramming
- \+ compile-time metaprogramming (AST transforms, **macros**, extension methods)
- \+ operator overloading
- \+ additional tools
- \+ additional productivity libraries
- \+ scripts & flexible language grammar

Macros

- macro, 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

```
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

- 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()
}
```

Macros

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

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

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

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

```
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

```
@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

What is Groovy?

```
Groovy = Java
  – boiler plate code
  + extensible dynamic and static natures
  + better functional programming
  + better OO programming
  + runtime metaprogramming
  + compile-time metaprogramming
  + operator overloading
  + additional tools (groovydoc, groovyConsole, groovysh)
  + additional productivity libraries
  + scripts & flexible language grammar
```

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

: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>
```

What is Groovy?

```
Groovy = Java
  – boiler plate code
  + extensible dynamic and static natures
  + better functional programming
  + better OO programming
  + runtime metaprogramming (extension methods)
  + compile-time metaprogramming (extension methods)
  + operator overloading
  + additional tools
  + additional productivity libraries
  + scripts & flexible language grammar
```

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

```
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"
    }
}
```

What is Groovy?

```
Groovy = Java
  – boiler plate code
  + extensible dynamic and static natures
  + better functional programming
  + better OO programming
  + runtime metaprogramming
  + compile-time metaprogramming
  + operator overloading
  + additional tools
  + additional productivity libraries
    (Regex, XML, SQL, JAXB, JSON, YAML, CLI, builders)
  + scripts & flexible language grammar
```

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:
```

CliBuilder supports annotations

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

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

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

JAXB marshalling shortcuts

```
@EqualsAndHashCode
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement
public class Person {
    String name
    int age
}
```

More concise without having to call createMarshaller() and createUnmarshaller()

```
JAXBContext jaxbContext = JAXBContext.newInstance(Person)
Person p = new Person(name: 'JT', age: 20)

String xml = jaxbContext.marshal(p)
assert jaxbContext.unmarshal(xml, Person) == p
```

A customizable JSON serializer

```
def generator = new JsonGenerator.Options()
        .addConverter(URL) { URL u, String key ->
    if (key == 'favoriteUrl') {
        u.getHost()
    } else {
        u
    }
}
.build()
```

What is Groovy?

```
Groovy = Java
  – boiler plate code
  + extensible dynamic and static natures
  + better functional programming
  + better OO programming
  + runtime metaprogramming
  + compile-time metaprogramming
  + operator overloading
  + additional tools
  + additional productivity libraries
  + scripts & flexible language grammar
```

Improved static type checking

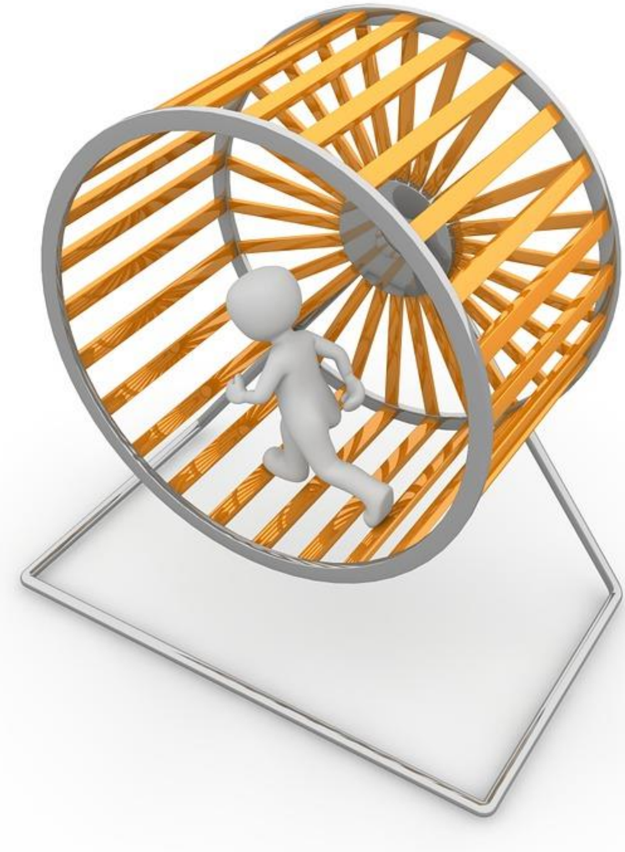
```
@CompileStatic
void testMultiAssigment() {
    def (String name, Integer age) = findPersonInfo()
    assert 'Daniel' == name
    assert 35 == age
}

@CompileStatic
Tuple2<String, Integer> findPersonInfo() {
    Tuple.tuple('Daniel', 35)
}
```

Groovy 3.0 Themes

- **Parrot parser**
  - **Improved copy/paste with Java**
  - **New syntax/operators**
- Indy by default
- JDK8 minimum and better JDK 9/10 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 9/10 JPMS support
- Additional documentation options



Catch up with Java

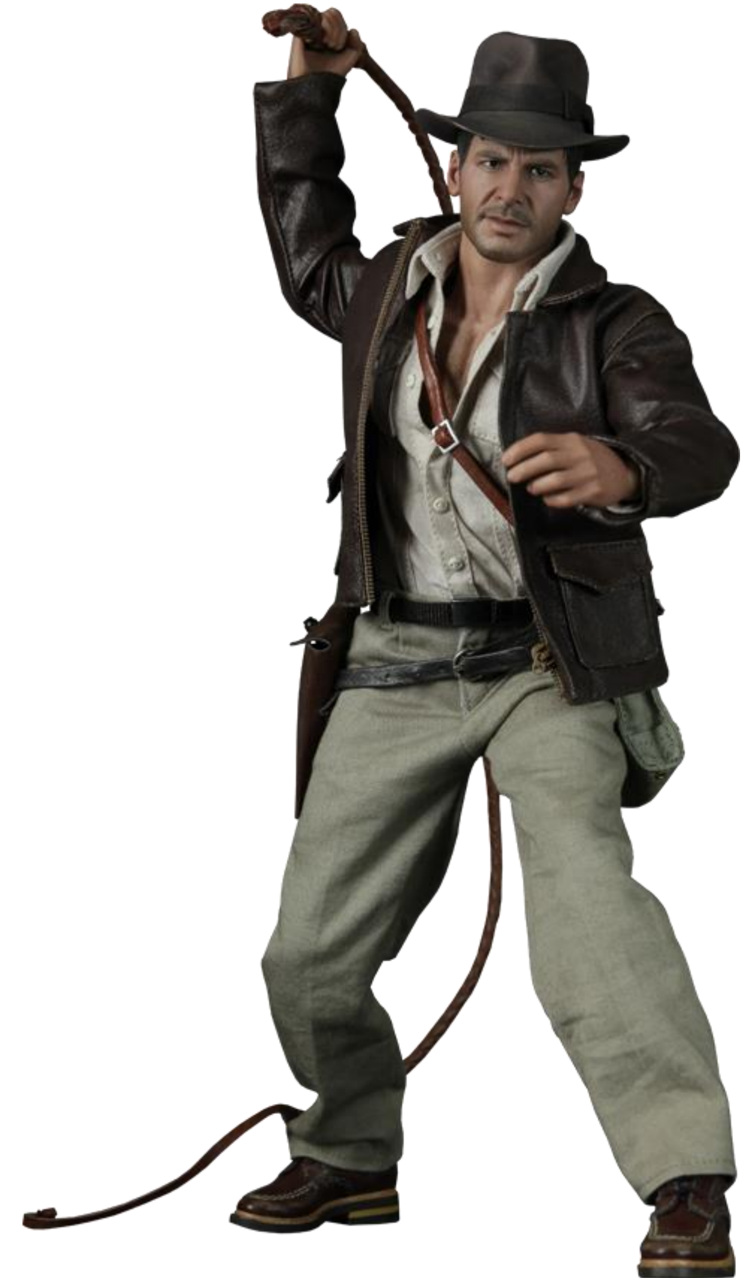
Groovy 3.0 Themes

- **Parrot parser**
  - **Improved copy/paste with Java**
  - New syntax/operators
- Indy by default
- JDK8 minimum and better JDK 9/10 JPMS support
- Additional documentation options

Java you will be assimilated: resistance is futile

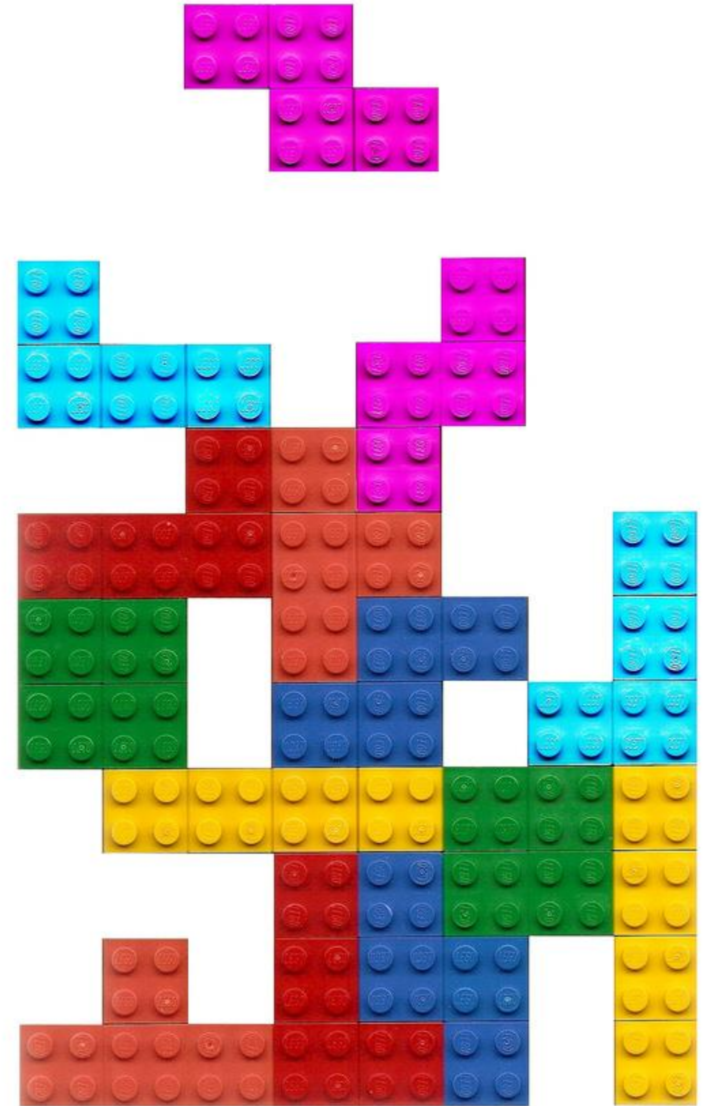
Groovy 3.0 Themes

- Parrot parser
  - Improved copy/paste with Java
  - New syntax/operators
- **Indy by default**
- JDK8 minimum and better JDK 9/10 JPMS support
- Additional documentation options



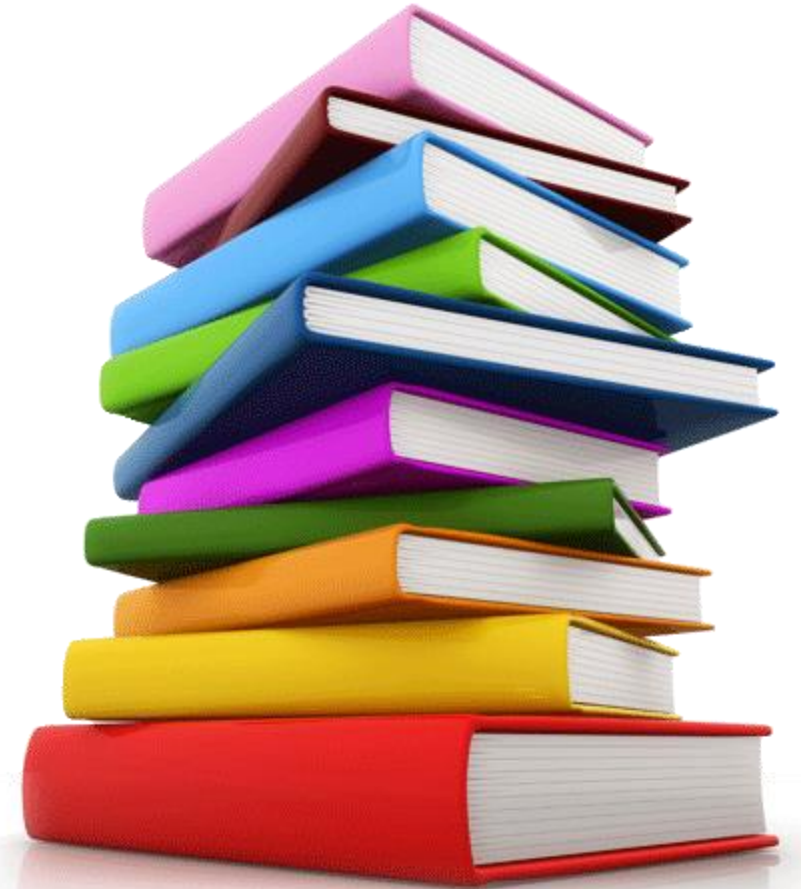
Groovy 3.0 Themes

- Parrot parser
  - Improved copy/paste with Java
  - New syntax/operators
- Indy by default
- **JDK8 minimum and better JDK 9/10 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 9/10 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

```groovy
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

```groovy
import groovy.transform.ToString

@ToString
class Element {
    String name
    int atomicNumber
}
def he = new Element(name: 'Helium')
he.with {
    // name = name != null ? name : 'Hydrogen'   // Java
    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: try with resources

```
// some Groovy friendliness without explicit types
def wrestle(s) {
    try (
            from = new FromResource(s)
            to = new ToResource()
    ) {
        to << from
        return to.toString()
    }
}

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

ToResource closing
FromResource closing

Better Java syntax support: try with resources

```
// some Groovy friendliness without explicit types
def wrestle(s) {
    try (
            from = new FromResource(s)
            to = new ToResource()
    ) {
        to << from
        return to.toString()
    }
}

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

But remember Groovy's IO/Resource extension methods may be better

ToResource closing
FromResource closing

Better Java syntax support: try with resources

```
def wrestle(s) {
    new FromResource(s).withCloseable { from ->
        new ToResource().withCloseable { to ->
            to << from
            to.toString()
        }
    }
}
```

But remember Groovy's IO/Resource extension methods may be better

ToResource closing
FromResource closing

Better Java syntax support: try with resources Java 9

```
def a = 1
def resource1 = new Resource(1)

try (resource1) {
    a = 2
}

assert Resource.closedResourceIds == [1]
assert 2 == a
```

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

```groovy
// 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
```

```groovy
// 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

```
// 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

```
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 {
    /**@ fo fum */
    def bar() { }
}

Foo.methods.find{ it.name == 'bar' }.groovydoc.content.contains('fo fum')
```

Requires: -Dgroovy.attach.runtime.groovydoc=true

Join us: groovy.apache.org