Why use Groovy in 2024?

Presented by
Dr Paul King
Unity Foundation &
VP Apache Groovy

EVERYTHINGOPEN GLADSTONE 2024

Gladstone, Australia, April 16-18, 2024

unityfoundation.io

Dr Paul King

Unity Foundation Groovy Lead

V.P. Apache Groovy

Author:

https://www.manning.com/books/groovy-in-action-second-edition

Slides:

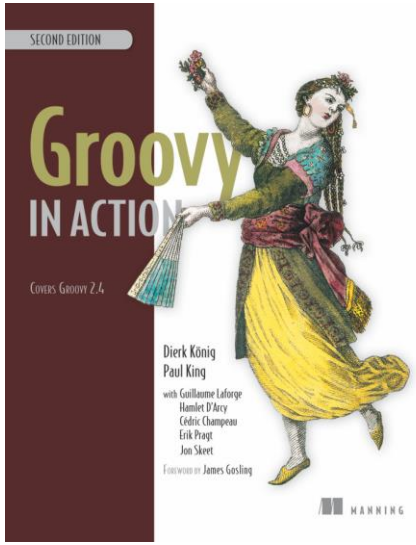
https://speakerdeck.com/paulk/groovy-today

Examples repo:

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

Twitter/X | Mastodon:

@paulk_asert | @paulk@foojay.social



Why use Groovy in 2024?

- Extension methods
  - *Improved out-of-the-box experience*
- Operator overloading
  - *Succinct code*
- AST transforms
  - *Reduced boilerplate and free design patterns*
- Weaker/Stronger typing
- Better:
  - *OO features, functional features, Scripting, non-stream aggregate processing*

Extension methods

- Conceptually *extend* a class with new methods
  - Instead of using a utility class like `StringUtils`

```
@Grab('org.apache.commons:commons-lang3:3.14.0')
import org.apache.commons.lang3.StringUtils

assert StringUtils.capitalize('foo') == 'foo'.capitalize()
```

Extension methods

- Conceptually *extend* a class with new methods
  - Instead of using a utility class like `StringUtils`

```
@Grab('org.apache.commons:commons-lang3:3.14.0')
import org.apache.commons.lang3.StringUtils

assert StringUtils.capitalize('foo') == 'foo'.capitalize()
```

  - The most common functionality, like `capitalize`, is built-in

Extension methods: almost 2000 across ~150 classes:

boolean[]
byte[]
char[]
double
double[]
double[][]
float
float[]
groovy.lang.Closure
groovy.lang.GString
groovy.lang.GroovyObject
groovy.lang.ListWithDefault
groovy.lang.MetaClass
groovy.sql.GroovyResultSet
int[]
int[][]
java.awt.Container
java.io.BufferedReader
java.io.BufferedWriter
java.io.Closeable
java.io.DataInputStream
java.io.File
java.io.InputStream
java.io.ObjectInputStream
java.io.ObjectOutputStream
java.io.OutputStream
java.io.PrintStream
java.io.PrintWriter
java.io.Reader
java.io.Writer
java.lang.Appendable
java.lang.AutoCloseable
java.lang.Boolean
java.lang.Byte[]
java.lang.CharSequence
java.lang.Character
java.lang.Class
java.lang.ClassLoader
java.lang.Comparable
java.lang.Double
java.lang.Enum
java.lang.Float
java.lang.Integer
java.lang.Iterable
java.lang.Long
java.lang.Number
java.lang.Object
java.lang.Object[]
java.lang.Process
java.lang.Runtime
java.lang.String
java.lang.StringBuffer
java.lang.StringBuilder
java.lang.String[]
java.lang.System
java.lang.System$Logger
java.lang.Thread
java.lang.Throwable
java.lang.reflect.AnnotatedElement
java.math.BigDecimal
java.math.BigInteger
java.net.ServerSocket
java.net.Socket
java.net.URL
java.nio.file.Path
java.sql.Date
java.sql.ResultSet
java.sql.ResultSetMetaData
java.sql.Timestamp
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.ZoneId
java.time.ZoneOffset
java.time.ZonedDateTime
java.time.chrono.ChronoPeriod
java.time.temporal.Temporal
java.time.temporal.TemporalAccessor
java.time.temporal.TemporalAmount
java.util.AbstractCollection
java.util.AbstractMap
java.util.BitSet
java.util.Calendar
java.util.Collection
java.util.Date
java.util.Deque
java.util.Enumeration
java.util.Iterator
java.util.List
java.util.Map
java.util.Optional
java.util.OptionalDouble
java.util.OptionalInt
java.util.OptionalLong
java.util.ResourceBundle
java.util.Set
java.util.SortedMap
java.util.SortedSet
java.util.Spliterator
java.util.TimeZone
java.util.Timer
java.util.concurrent.BlockingQueue
java.util.concurrent.Future
java.util.regex.Matcher
java.util.regex.Pattern
java.util.stream.BaseStream
java.util.stream.Stream
javax.script.ScriptEngine
javax.script.ScriptEngineManager
javax.swing.AbstractButton
javax.swing.ButtonGroup
javax.swing.DefaultComboBoxModel
javax.swing.DefaultListModel
javax.swing.JComboBox
javax.swing.JMenu
javax.swing.JMenuBar
javax.swing.JPopupMenu
javax.swing.JTabbedPane
javax.swing.JToolBar
javax.swing.ListModel
javax.swing.MutableComboBoxModel
javax.swing.table.DefaultTableModel
javax.swing.table.TableColumnModel
javax.swing.table.TableModel
javax.swing.tree.DefaultMutableTreeNode
javax.swing.tree.MutableTreeNode
javax.swing.tree.TreeNode
javax.swing.tree.TreePath
long
long[]
long[][]
org.codehaus.groovy.ast.ASTNode
org.codehaus.groovy.control.SourceUnit
org.codehaus.groovy.macro.matcher.ASTMatcher
org.codehaus.groovy.macro.runtime.MacroContext
org.codehaus.groovy.runtime.NullObject
org.w3c.dom.Element
org.w3c.dom.NodeList
short[]

Primitive array extension methods

```
class ArrayMax {
    private static IntComparator maxAbs
        = (i, j) -> i.abs() <=> j.abs()

    static int max(int[] nums) { nums.max() }

    static int maxAbs(int[] nums) { nums.max(maxAbs) }
}
```

```
class StreamsMax {
    private static Comparator<Integer> maxAbs =
        Comparator.<Integer>comparingInt(Math::abs)

    static int max(int[] nums) {
        nums.intStream().max().getAsInt()
    }

    static int maxAbs(int[] nums) {
        nums.stream().max(maxAbs).get()
    }
}
```

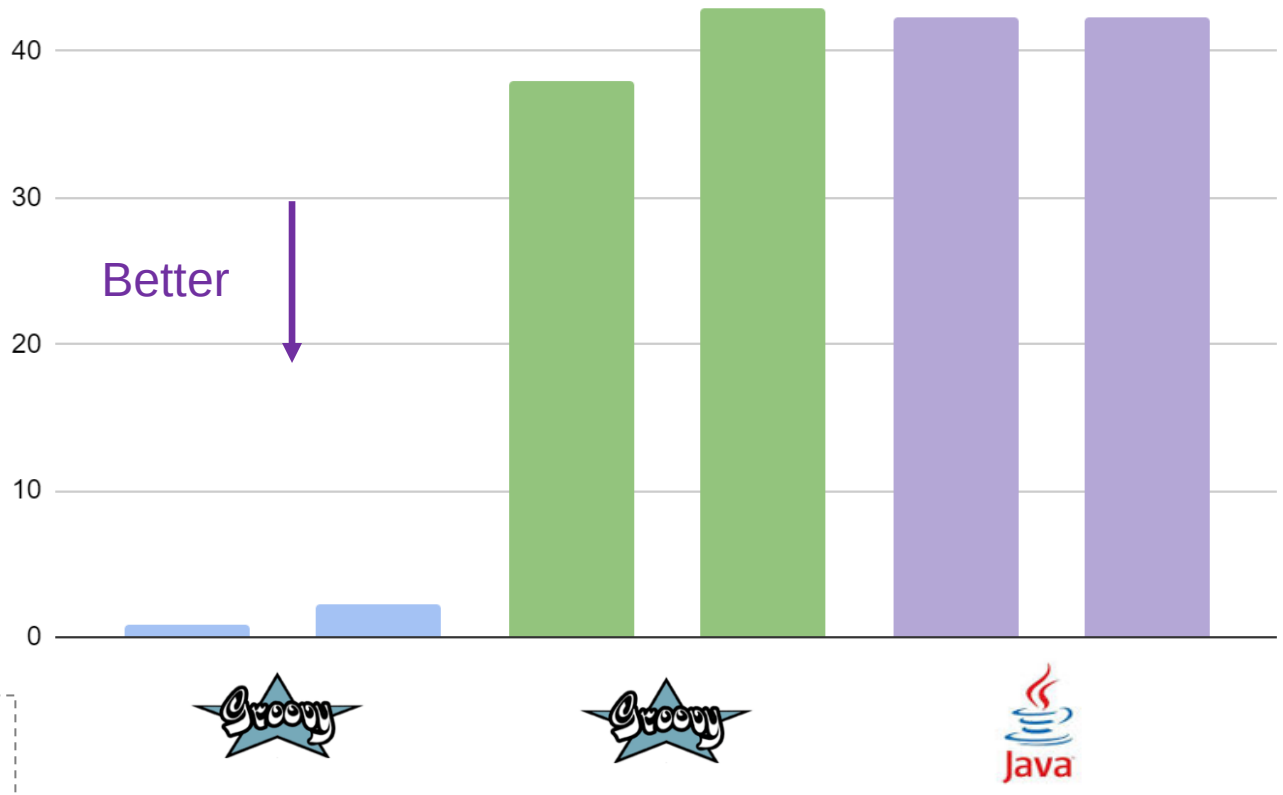
Primitive array extension methods

```groovy
class ArrayMax {
    private static IntComparator maxAbs
        = (i, j) -> i.abs() <=> j.abs()

    static int max(int[]

    static int maxAbs(i
}
```

```groovy
class StreamsM
    private s
        Compa

    static in
        nums.
    }

    static int maxAbs(int[] nums) {
        nums.stream().max(maxAbs).get()
    }
}
```

```java
public class JavaStreamsMax {
    private static Comparator<Integer> comparator
        = Comparator.comparingInt(Math::abs);

    public static int max(int[] nums) {
        return Arrays.stream(nums).max().getAsInt();
    }

    public static int maxAbs(int[] nums) {
        return Arrays.stream(nums).boxed().max(comparator).get();
    }
}
```

Primitive array extension methods




```
int[] numbers = {10, 20, 15, 30, 5};
```

Benchmark	Mode	Cnt	Score	Units
ArrayStreamsBenchmark.arrayMax	avgt	10	0.779	ns/op
ArrayStreamsBenchmark.arrayMaxAbs	avgt	10	2.302	ns/op
ArrayStreamsBenchmark.streamsMax	avgt	10	38.024	ns/op
ArrayStreamsBenchmark.streamsMaxAbs	avgt	10	43.100	ns/op
ArrayStreamsBenchmark.javaStreamsMax	avgt	10	42.262	ns/op
ArrayStreamsBenchmark.javaStreamsMaxAbs	avgt	10	42.346	ns/op

Extension methods

- For dynamic and static modes
  - *Conventions allow IDE discovery*

```
class FileExtensionMethods {
    static int getWordCount(File self) {
        self.text.split(/\w+/).size()
    }
}
```

```
file.getWordCount()
file.wordCount
```

Operator Overloading

Operator	Method	Operator	Method
`+`	*a.plus(b)*	`a[b]`	*a.getAt(b)*
`-`	*a.minus(b)*	`a[b] = c`	*a.putAt(b, c)*
`*`	*a.multiply(b)*	`a in b`	*b.isCase(a)*
`/`	*a.div(b)*	`<<`	*a.leftShift(b)*
`%`	*a.mod(b)* *a.remainder(b)*	`>>`	*a.rightShift(b)*
		`>>>`	*a.rightShiftUnsigned(b)*
`**`	*a.power(b)*	`==>`	*a.implies(b)*
`==`	*a.equals(b)*	`<=>`	*a.compareTo(b)*
`|`	*a.or(b)*	`++`	*a.next()*
`&`	*a.and(b)*	`--`	*a.previous()*
`^`	*a.xor(b)*	`+a`	*a.positive()*
`as`	*a.asType(b)*	`-a`	*a.negative()*
`a()`	*a.call()*	`~a`	*a.bitwiseNegate()*

Operator Overloading

BigInteger

```java
assertEquals(BigInteger.valueOf(21),
    BigInteger.valueOf(12).add(BigInteger.valueOf(9)));
```


```groovy
assert 21G == 12G + 9G
```

Matrices

org.apache.commons:commons-math3:3.6.1

```java
assertEquals(m3, m1.multiply(m2.power(2)));
```


```groovy
assert m3 == m1 * m2 ** 2
```

Operator Overloading

OperatorRename in Groovy 5

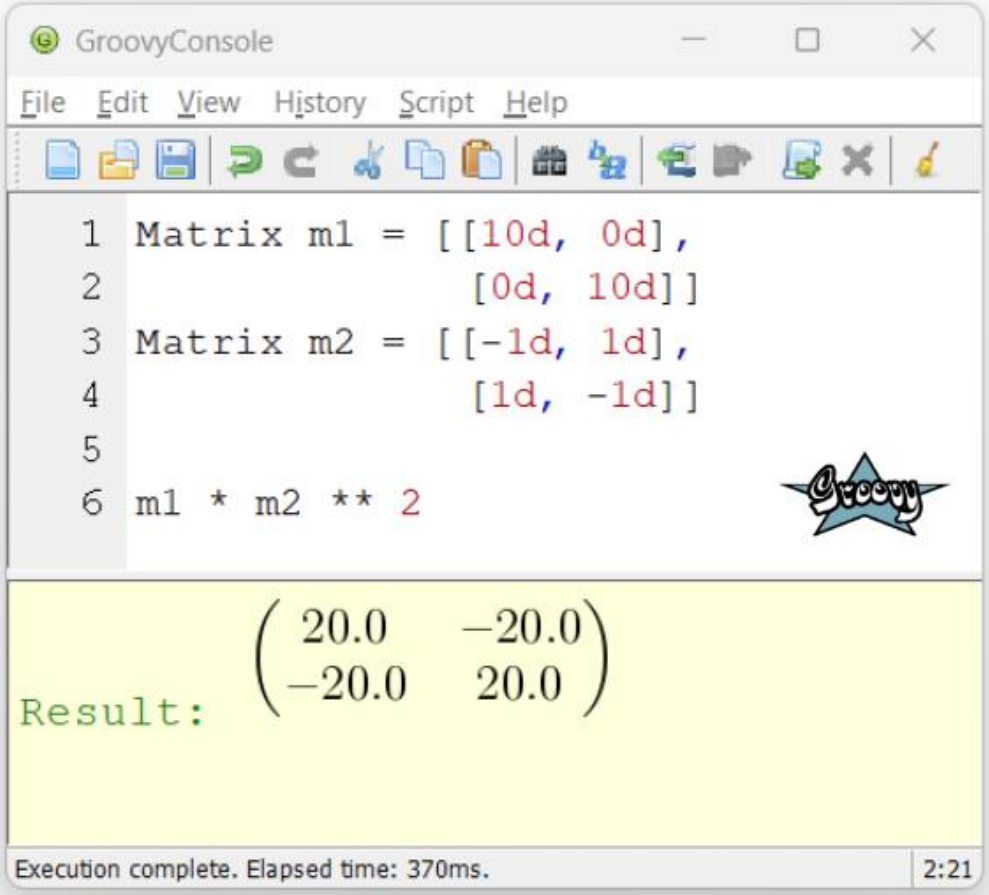
```
@OperatorRename(plus='add', multiply='scalarMultiply')
def testMatrixOperations() {
    double[][] d = [ [1d, 0d], [0d, 1d] ]
    var m = MatrixUtils.createRealMatrix(d)
    assert m.add(m) == m.scalarMultiply(2)    // methods unchanged
    assert m + m == m * 2                     // additional operator mappings
}
```

Operator Overloading

```
jshell> import org.apache.commons.math3.linear.MatrixUtils

jshell> double[][] d1 = { {10d, 0d}, {0d, 10d}}
d1 ==> double[2][] { double[2] { 10.0, 0.0 }, double[2] { 0.0, 10.0 } }

jshell> var m1 = MatrixUtils.createRealMatrix(d1)
m1 ==> Array2DRowRealMatrix{{10.0,0.0},{0.0,10.0}}

jshell> double[][] d2 = { {-1d, 1d}, {1d, -1d}}
d2 ==> double[2][] { double[2] { -1.0, 1.0 }, double[2] { 1.0, -1.0 } }

jshell> var m2 = MatrixUtils.createRealMatrix(d2)
m2 ==> Array2DRowRealMatrix{{-1.0,1.0},{1.0,-1.0}}

jshell> System.out.println(m1.multiply(m2.power(2)))
Array2DRowRealMatrix{{20.0,-20.0},{-20.0,20.0}}
```

Operator Overloading (plus other features)

```
jshell> import org.apache.commons.math3.linear.MatrixUtils

jshell> double[][] d1 = { {10d, 0d},
d1 ==> double[2][] { double[2] { 10.

jshell> var m1 = MatrixUtils.createR
m1 ==> Array2DRowRealMatrix{{10.0,0.

jshell> double[][] d2 = { {-1d, 1d},
d2 ==> double[2][] { double[2] { -1.

jshell> var m2 = MatrixUtils.createR
m2 ==> Array2DRowRealMatrix{{-1.0,1.

jshell> System.out.println(m1.multip
Array2DRowRealMatrix{{20.0,-20.0},{-20.0,20.0}}
```



Other dynamic features: Adding methods at runtime

```
File.metaClass.getWordCount = {
    delegate.text.split(/\w+/).size()
}

String.metaClass.getResource = {
    new File(getClass().classLoader.getResource(delegate).toURI())
}

assert 'magna_carta_latin.txt'.resource.wordCount == 3771
assert 'magna_carta_en.txt'.resource.wordCount == 4740
```

Other dynamic features: Lifecycle hooks

```
class Foo {
    def methodMissing(String name, args) {
        "You called $name(${args.join(', ')})"
    }
}

var foo = new Foo()
assert foo.unknown() == 'You called unknown()'
assert foo.divide(0) == 'You called divide(0)'
assert foo.add(1, 2) == 'You called add(1, 2)'
```

Other dynamic features: Dangling closure builder pattern

```
def writer = new StringWriter()
def pom = new MarkupBuilder(writer)

pom.project {
    modelVersion('4.0.0')
    groupId('org.apache.groovy')
    artifactId('groovy-examples')
    version('1.0-SNAPSHOT')
}

assert writer.toString() == '''\
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.apache.groovy</groupId>
  <artifactId>groovy-examples</artifactId>
  <version>1.0-SNAPSHOT</version>
</project>'''
```

AST Transformations

```
@Immutable(copyWith = true)
@Sortable(excludes = 'authors')
@AutoExternalize
class Book {
    @IndexedProperty
    List<String> authors

    String title

    Date publicationDate
}
```

AST Transformations

```
// imports not shown
public class Book {

    private String $to$string;
    private int $hash$code;
    private final List<String> authors;
    private final String title;
    private final Date publicationDate;
    private static final java.util.Comparator this$TitleComparator;
    private static final java.util.Comparator this$PublicationDateComparator;

    public Book(List<String> authors, String title, Date publicationDate) {
        if (authors == null) {
            this.authors = null;
        } else {
            if (authors instanceof Cloneable) {
                List<String> authorsCopy = (List
                this.authors = (List<String>) (
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : authorsCopy instanceof
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : authorsCopy instanceof
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : authorsCopy instanceof
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : authorsCopy instanceof
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : DefaultGroovyMethods
            } else {
                this.authors = (List<String>) (
DefaultGroovyMethods.asImmutable(authors)
                        : authors instanceof So
DefaultGroovyMethods.asImmutable(authors)
                        : authors instanceof Se
DefaultGroovyMethods.asImmutable(authors)
                        : authors instanceof Ma
DefaultGroovyMethods.asImmutable(authors)
                        : authors instanceof Li
DefaultGroovyMethods.asImmutable(authors)
                        : DefaultGroovyMethods
            }
        }
        this.title = title;
        if (publicationDate == null) {
            this.publicationDate = null;
        } else {
            this.publicationDate = (Date) publi
        }
    }

    public Book(Map args) {
        if ( args == null) {
            args = new HashMap();
        }
        ImmutableASTTransformation.checkPropNa
        if (args.containsKey("authors")) {
            if ( args.get("authors") == null) {
                this .authors = null;
            } else {
                if (args.get("authors") instanc
                    List<String> authorsCopy = (List<String>) ((ArrayList<?>)
args.get("authors")).clone();
                    this.authors = (List<String>) (authorsCopy instanceof SortedSet ?
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : authorsCopy instanceof SortedMap ?
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : authorsCopy instanceof Set ?
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : authorsCopy instanceof Map ?
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : authorsCopy instanceof List ?
DefaultGroovyMethods.asImmutable(authorsCopy)
                        : DefaultGroovyMethods.asImmutable(authorsCopy));
                } else {
                    List<String> authors = (List<String>) args.get("authors");
                    this.authors = (List<String>) (authors instanceof SortedSet ?
DefaultGroovyMethods.asImmutable(authors)
                        : authors instanceof SortedMap ?
DefaultGroovyMethods.asImmutable(authors)
                        : authors instanceof Set ?
DefaultGroovyMethods.asImmutable(authors)
                        : authors instanceof Map ?
```

```
    public Book() {
        this (new HashMap());
    }

    public int compareTo(Book other) {
        if (this == other) {
            return 0;
        }
        Integer value = 0
        value = this .title <=> other .title
        if ( value != 0) {
            return value
        }
```

```
@Immutable(copyWith = true)
@Sortable(excludes = 'authors')
@AutoExternalize
class Book {
    @IndexedProperty
    List<String> authors

    String title

    Date publicationDate
}
```

```
            if (!(this.getAuthors().equals(this))) {
                _result = HashCodeHelper.updateHash(_result,
this.getAuthors());
            }
            if (!(this.getTitle().equals(this))) {
                _result = HashCodeHelper.updateHash(_result,
this.getTitle());
            }
            if (!(this.getPublicationDate().equals(this))) {
                _result = HashCodeHelper.updateHash(_result,
this.getPublicationDate());
            }
            $hash$code = (int) _result;
        }
        return $hash$code;
    }

    public boolean canEqual(Object other) {
        return other instanceof Book;
    }
```

```
    public boolean equals(Object other) {
        if ( other == null) {
            return false;
        }
        if (this == other) {
            return true;
        }
        if (!( other instanceof Book)) {
            return false;
        }
        Book otherTyped = (Book) other;
        if (!(otherTyped.canEqual( this ))) {
            return false;
        }
        if (!(this.getAuthors() == otherTyped.getAuthors())) {
            return false;
        }
        if (!(this.getTitle().equals(otherTyped.getTitle()))) {
            return false;
        }
        if
!(this.getPublicationDate().equals(otherTyped.getPublicationDate()))
 {
            return false;
        }
        return true;
    }

    public final Book copyWith(Map map) {
        if (map == null || map.size() == 0) {
            return this;
        }
        Boolean dirty = false;
        HashMap construct = new HashMap();
        if (map.containsKey("authors")) {
            Object newValue = map.get("authors");
            Object oldValue = this.getAuthors();
            if (newValue != oldValue) {
                oldValue = newValue;
                dirty = true;
            }
            construct.put("authors", oldValue);
        } else {
            construct.put("authors", this.getAuthors());
        }
        if (map.containsKey("title")) {
            Object newValue = map.get("title");
            Object oldValue = this.getTitle();
            if (newValue != oldValue) {
                oldValue = newValue;
                dirty = true;
            }
            construct.put("title", oldValue);
        } else {
            construct.put("title", this.getTitle());
        }
        if (map.containsKey("publicationDate")) {
            Object newValue = map.get("publicationDate");
            Object oldValue = this.getPublicationDate();
            if (newValue != oldValue) {
                oldValue = newValue;
                dirty = true;
            }
            construct.put("publicationDate", oldValue);
        } else {
            construct.put("publicationDate",
this.getPublicationDate());
        }
        return dirty == true ? new Book(construct) : this;
    }

    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(authors);
        out.writeObject(title);
        out.writeObject(publicationDate);
    }

    public void readExternal(ObjectInput oin) throws IOException,
ClassNotFoundException {
        authors = (List) oin.readObject();
        title = (String) oin.readObject();
        publicationDate = (Date) oin.readObject();
```

```
    static {
        this$TitleComparator = new Book$TitleComparator();
        this$PublicationDateComparator = new
Book$PublicationDateComparator();
    }

    public String getAuthors(int index) {
        return authors.get(index);
    }

    public List<String> getAuthors() {
        return authors;
    }

    public final String getTitle() {
        return title;
    }

    public final Date getPublicationDate() {
        if (publicationDate == null) {
            return publicationDate;
        } else {
            return (Date) publicationDate.clone();
        }
    }

    public int compare(java.lang.Object param0, java.lang.Object
param1) {
        return -1;
    }

    private static class Book$TitleComparator extends
AbstractComparator<Book> {
        public Book$TitleComparator() {
        }

        public int compare(Book arg0, Book arg1) {
            if (arg0 == arg1) {
                return 0;
            }
            if (arg0 != null && arg1 == null) {
                return -1;
            }
            if (arg0 == null && arg1 != null) {
                return 1;
            }
            return arg0.title <=> arg1.title;
        }

        public int compare(java.lang.Object param0, java.lang.Object
param1) {
            return -1;
        }

    }

    private static class Book$PublicationDateComparator extends
AbstractComparator<Book> {

        public Book$PublicationDateComparator() {
        }

        public int compare(Book arg0, Book arg1) {
            if ( arg0 == arg1 ) {
                return 0;
            }
            if ( arg0 != null && arg1 == null) {
                return -1;
            }
            if ( arg0 == null && arg1 != null) {
                return 1;
            }
            return arg0 .publicationDate <=> arg1 .publicationDate;
        }


        public int compare(java.lang.Object param0, java.lang.Object
param1) {
            return -1;
        }

    }
```

AST Transformations

```groovy
@Immutable(copyWith = true)
@Sortable(excludes = 'authors')
@AutoExternalize
class Book {
    @IndexedProperty
    List<String> authors

    String title

    Date publication Date
}
```

10 lines of Groovy
or
600 lines of Java

AST Transformations: Groovy 2.4, 2.5, 2.5 (improved), 3.0, 4.0, 5.0

- 80 AST transforms

@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
@NullCheck

@OperatorRename

@NonSealed
@RecordBase
@Sealed
@PlatformLog
@GQ
@Final
@RecordType
@POJO
@Pure
@Contracted
@Ensures
@Invariant
@Requires
@ClassInvariant
@ContractElement
@Postcondition
@Precondition

Other static features: extensible type checker

```
def newYearsEve = '2020-12-31'
def matcher = newYearsEve =~ /(\d{4})-(\d{1,2})-(\d{1,2}/  // PatternSyntaxException
```

```
import groovy.transform.TypeChecked

@TypeChecked(extensions = 'groovy.typecheckers.RegexChecker')
def whenIs2020Over() {
    def newYearsEve = '2020-12-31'
    def matcher = newYearsEve =~ /(\d{4})-(\d{1,2})-(\d{1,2}/
}
```

```
1 compilation error:
[Static type checking] - Bad regex: Unclosed group near index 26
(\d{4})-(\d{1,2})-(\d{1,2}
 at line: 6, column: 19
```

Other features

- Ranges
  - `1..5, 'a'<..<'c'`
- Default parameters
  - `def coords(x, y = -1, z = 0) { }`
- Named arguments
  - `hypotenuse(x: 3, y: 4, z: 5)`
- Command chains
  - `move right by 2.m at 5.cm/s`
  - `please show the square_root of 100`
  - `まず 100 の の 平方根 を 表示する`
- Groovy Language INtegrated Queries (Ginq/Gquery)

```
from p in persons
leftjoin c in cities on p.city.name == c.name
where c.name == 'Shanghai'
select p.name, c.name as cityName
```

Case Study: Roman Numerals

```
I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000
```

- The value of a symbol is added to itself, as many times as it appears
- A symbol can be repeated only three times
- V, L, and D are never repeated
- When a symbol of smaller value appears after a symbol of greater value, its values will be added
- When a symbol of a smaller value appears before a greater value symbol, it will be subtracted

Case Study: Roman Numerals

com.github.fracpete:romannumerals4j:0.0.1

Direct library usage

```
var fmt = new RomanNumeralFormat()
assert fmt.parse('XII') == 12
assert fmt.format(9) == 'IX'
assert fmt.format(fmt.parse('XII')
    + fmt.parse('IX')) == 'XXI'
```

Case Study: Roman Numerals

Using dynamic metaprogramming

```
String.metaClass.toDecimal {
    fmt.parse(delegate)
}
Integer.metaClass.toRoman {
    fmt.format(delegate)
}
```

```
assert 'XII'.toDecimal() == 12
assert 9.toRoman() == 'IX'
assert 'XII'.toDecimal() + 'IX'.toDecimal() == 'XXI'.toDecimal()
```

Case Study: Roman Numerals

Using extension methods

```
static Integer toDecimal(String self) {
    fmt.parse(self)
}
static String toRoman(Integer self) {
    fmt.format(self)
}
```

```
assert 'XII'.toDecimal() == 12
assert 9.toRoman() == 'IX'
assert 'XII'.toDecimal() + 'IX'.toDecimal() == 'XXI'.toDecimal()
```

Case Study: Roman Numerals

With a domain class

```
class RomanNumeral {
    private fmt = new RomanNumeralFormat()
    private int d
    RomanNumeral(String s) { d = fmt.parse(s) }
    RomanNumeral(int d) { this.d = d }
    def plus(RomanNumeral other) { new RomanNumeral(d + other.d) }
    String toString() { fmt.format(d) }
    boolean equals(other) { d == other.d }
}
```

Case Study: Roman Numerals

And a lifecycle method hook:

```
def propertyMissing(String p) {
    new RomanNumeral(p)
}
```

Gives improved coding experience:

```
assert XII + IX == XXI
```

Case Study: Roman Numerals

```
class RomanNumeral implements Comparable {
    private fmt = new RomanNumeralFormat()
    private int d
    RomanNumeral(String s) { d = fmt.parse(s) }
    RomanNumeral(int d) { this.d = d }
    def plus(RomanNumeral other) { new RomanNumeral(d + other.d) }
    def multiply(RomanNumeral other) { new RomanNumeral(d * other.d) }
    String toString() { fmt.format(d) }
    int compareTo(other) { d <=> other.d }
    boolean equals(other) { d == other.d }
    RomanNumeral next() { new RomanNumeral(d+1) }
}
```

Case Study: Roman Numerals

```
assert XII + IX == XXI

assert [LVII + LVII, V * III, IV..(V+I)]
    == [       CXIV,      XV,    IV..VI]

assert switch(L) {
    case L        -> '50 exactly'
    case XLV..LV -> 'close to 50'
    default       -> 'not close to 50'
} == '50 exactly'
```

Case Study: Roman Numerals

```
@Sortable @Canonical
class RomanNumeral {
    private fmt = new RomanNumeralFormat()
    final int d
    RomanNumeral(String s) { d = fmt.parse(s.toUpperCase()) }
    RomanNumeral(int d) { this.d = d }
    def plus(RomanNumeral other) { new RomanNumeral(d + other.d) }
    def multiply(RomanNumeral other) { new RomanNumeral(d * other.d) }
    String toString() { fmt.format(d) }
    RomanNumeral next() { new RomanNumeral(d+1) }
}
```

```
assert [X, IX, IV, V, VI].sort() == [iv, v, vi, ix, x]
```

Case Study: Roman Numerals

3999 is the biggest roman numeral, otherwise we violate the rule about never having more than 3 of the same character in succession, so this statement:

```
assert MMMCMXCIX + I == MMMM
```

Intentionally gives this runtime error:

```
Caught: java.text.ParseException: Unparseable number: "MMMM"
```

But we can use type checking and add a custom type checking extension …

Case Study: Roman Numerals

```
unresolvedVariable { VariableExpression ve ->
    try {
        new RomanNumeral(ve.name)
        storeType(ve, classNodeFor(RomanNumeral))
    } catch(ParseException unused) {
        addStaticTypeError("Not a valid roman numeral: $ve.name", ve)
    }
    handled = true
}
```

Now we get this compilation error:

```
[Static type checking] - Not a valid roman numeral: MMMM
 @ line 6, column 25.
   assert MMMCMXCIX + I == MMMM
                           ^
```

Better OO features

- Traits

Traits

- Can be like Java default interface methods

```java
import java.util.Collections;
import java.util.List;

public interface RotatableList<E> extends List<E> {
    default void rotate(int distance) {
        Collections.rotate(this, distance);
    }
}
```

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

public class RotatableListImpl extends ArrayList<String>
        implements RotatableList<String> {
    public RotatableListImpl() {
        super(List.of("p", "i", "n", "s"));
    }
}
```

```java
public class RotateMain {
    public static void main(String[] args) {
        var myList = new RotatableListImpl();
        myList.rotate(1);
        System.out.println(myList);
    }
}
```

```
[s, p, i, n]
```

Traits

- Can be like Java default interface methods

```
trait RotatableList<E> implements List<E> {
    void rotate(int distance) {
        Collections.rotate(this, distance)
    }
}

class RotatableListImpl extends ArrayList<String> implements RotatableList<String> {
    RotatableListImpl() { super(['p', 'i', 'n', 's']) }
}

var myList = new RotatableListImpl()
myList.rotate(1)
assert myList == ['s', 'p', 'i', 'n']
```

Traits

- But traits are more ambitious, supporting:
  - *Sharing state information (stateful traits)*
  - *More flexible selection of functionality*
    - *Sharable behavior not just default behavior*
    - *An OO feature not just primarily about API evolution*
  - *Traits at runtime (dynamic traits)*
  - *Mixin/Stackable traits pattern (stackable traits)*

Traits: Stateful Traits

```
trait Fighter {
    int health
    int strength
    void fight(Fighter enemy) {
        health -= enemy.strength
        enemy.health -= strength
        if (health <= 0) println "$name has lost"
        if (enemy.health <= 0) println "$enemy.name was defeated"
    }
}

trait HasName {
    String name
}
```

Traits: Stateful Traits

```
trait Fighter {
    int health
    int strength
    void fight(Fighter
        health -= enemy
        enemy.health -=
        if (health <= 0
```

```
@TupleConstructor(allProperties=true)
class Player implements Fighter, HasName { }

@TupleConstructor(allProperties=true)
class Boss implements Fighter, HasName { }

var p1 = new Player('Bowser', 75, 40)
var p2 = new Player('Mario', 65, 30)
var p3 = new Player('Sonic', 55, 35)
var boss = new Boss('Giga Bowser’, 100, 50)
p1.fight(boss)
p2.fight(boss)
p3.fight(boss)
```

Giga Bowser was defeated

Traits: Dynamic Traits

```
trait Starable {
    String starify() {
        this.replaceAll('o', '⭐')
    }
}

def groovy = 'Groovy' as Starable
assert groovy.starify() == 'Gr⭐⭐vy'
```

Traits: More flexible behavior sharing

- Groovy normalizes negative index values when using the index notation or `getAt` method, but not the `get` method

```
def nums = [1, 2, 3]
assert nums[-1] == 3
assert nums.getAt(-1) == 3

shouldFail(IndexOutOfBoundsException) {
    nums.get(-1)
}
```

- What if you also wanted this for the `get` method?

Traits: More flexible behavior sharing

- Index normalization for the `get` method

```
trait NormalizedGet<E> implements List<E> {
    E get(int index) {
        if (index < 0) index += size()
        super.get(index)
    }
}

class MyList extends ArrayList implements NormalizedGet {}

nums = [1, 2, 3] as MyList
assert nums.get(-1) == 3
```

Traits: Stackable Traits Pattern

```
interface Handler {
    String handle(String message)
}

trait UpperHandler implements Handler {
    String handle(String message) { super.handle(message.toUpperCase()) }
}

trait ReverseHandler implements Handler {
    String handle(String message) { super.handle(message.reverse()) }
}

trait StarHandler implements Handler {
    String handle(String message) { message.replaceAll('O', '⭐') }
}

class MyHandler implements StarHandler, ReverseHandler, UpperHandler {}

assert new MyHandler().handle('yvoorG') == 'GR⭐⭐VY'
```

Better functional features

- Tail recursion
- Memoization

Closures

- Somewhat like Lambdas
  - but let's explore some recursion examples

```
import java.math.BigInteger;
import java.util.function.UnaryOperator;

public class FactLambda {
    static UnaryOperator<BigInteger> factorial;
    public static void main(String[] args) {
        factorial = n -> n.equals(BigInteger.ZERO)
                ? BigInteger.ONE
                : n.multiply(factorial.apply(n.subtract(BigInteger.ONE)));
        System.out.println(factorial.apply(BigInteger.valueOf(5)));    // 120
        System.out.println(factorial.apply(BigInteger.valueOf(8000))); // Boom!
    }
}
```

StackOverflowError

Closures: Naïve algorithm

```
def factorial
factorial = {
    it <= 1 ? 1G : it * factorial(it - 1)
}

println factorial(5)    // 120
println factorial(8000) // StackOverFlow
```

StackOverflowError

```
factorial(5)
5 * factorial(4)
5 * (4 * factorial(3))
5 * (4 * (3 * factorial(2)))
5 * (4 * (3 * (2 * factorial(1))))
5 * (4 * (3 * (2 * 1)))
5 * (4 * (3 * 2))
5 * (4 * 6)
5 * 24
120
```

Closures: Tail recursive algorithm

```
def factorial
factorial = { n, acc = 1G ->
    n <= 1 ? acc : factorial(n - 1, n * acc)
}

println factorial(5)    // 120
println factorial(8000) // StackOverFlow
```

StackOverflowError

```
factorial(5, 1)
  factorial(4, 5)
    factorial(3, 20)
      factorial(2, 60)
        factorial(1, 120)
120
```

Closures: Tail recursive with trampoline

```
def factorial
factorial = { n, acc = 1G ->
    n <= 1 ? acc : factorial.trampoline(n - 1, n * acc)
}.trampoline()

println factorial(5)
println factorial(8000)
println factorial(100_000)
```

```
120
5184…<27750 more digits>
28242…<456570 more digits>
```

Closures: Tail recursive with trampoline

```
def factorial
factorial = { n, acc = 1G ->
    n <= 1 ? acc : factorial.trampoline(n - 1, n * acc)
}.trampoline()

println factorial(5)
println factorial(8000)
println factorial(100_000)
```

```
120
5184…<27750 more digits>
28242…<456570 more digits>
```

See also @TailRecursive

Closures: Memoize

```java
import java.math.BigInteger;
import java.util.function.UnaryOperator;

public class FiboLambda {
    static UnaryOperator<Integer> fibI;
    static UnaryOperator<Long> fibL;
    static UnaryOperator<BigInteger> fibBI;
    public static void main(String[] args) {
        fibI = n -> n <= 1 ? n : fibI.apply(n - 1) + fibI.apply(n - 2);
        fibL = n -> n <= 1 ? n : fibL.apply(n - 1) + fibL.apply(n - 2);
        fibBI = n -> n.compareTo(BigInteger.ONE) <= 0
                ? N
                : fibBI.apply(n.subtract(BigInteger.ONE)).add(fibBI.apply(n.subtract(BigInteger.TWO)));

        var start = System.currentTimeMillis();
        System.out.println(fibI.apply(10)); // 55
        System.out.println(fibL.apply(50L)); // 12586269025
        System.out.println(fibBI.apply(BigInteger.valueOf(100))); // 354224848179261915075
        var end = System.currentTimeMillis();
        var years = (end - start) / 1000 / 60 / 60.0 / 24 / 365.25;
        System.out.println("Completed in " + years + " years");
    }
}
```

Closures: Memoize

```
import java.math.BigInteger;
import java.util.function.UnaryOperator;

public class FiboLambda {
    static UnaryOperator<Integer> fibI;
    static UnaryOperator<Long> fibL;
    static UnaryOperator<BigInteger> fibBI;
    public static void main(String[] args) {
        fibI = n -> n <= 1 ? n : fibI.apply(n - 1) + fibI.apply(n - 2);
        fibL = n -> n <= 1 ? n : fibL.apply(n - 1) + fibL.apply(n - 2);
        fibBI = n -> n.compareTo(BigInteger.ONE) <= 0
                ? N
                : fibBI.apply(n.subtract(BigInteger.ONE)).add(fibBI.apply

        var start = System.currentTimeMillis();
        System.out.println(fibI.apply(10)); // 55
        System.out.println(fibL.apply(50L)); // 12586269025
        System.out.println(fibBI.apply(BigInteger.valueOf(100))); // 354224848179261915075
        var end = System.currentTimeMillis();
        var years = (end - start) / 1000 / 60 / 60.0 / 24 / 365.25;
        System.out.println("Completed in " + years + " years");
    }
}
```

```
55
12586269025
354224848179261915075
Completed in 6.4E9 years*
```

**** Estimated Fibonacci time complexity has $O(2^n)$ as the upper bound***

Closures: Memoize

```
var fib
fib = { n -> n <= 1 ? n : fib(n - 1) + fib(n - 2) }

var start = System.currentTimeMillis()
assert fib(10) == 55
assert fib(50L) == 12586269025L
assert fib(100G) == 354224848179261915075G
println "Completed in ${System.currentTimeMillis() - start} ms"
```

Closures: Memoize

```
Completed in 117 ms
```

```
var fib
fib = { n -> n <= 1 ? n : fib(n - 1) + fib(n - 2) }.memoize()

var start = System.currentTimeMillis()
assert fib(10) == 55
assert fib(50L) == 12586269025L
assert fib(100G) == 354224848179261915075G
println "Completed in ${System.currentTimeMillis() - start} ms"
```

Closures: Memoize

```
Completed in 117 ms
```

```
var fib
fib = { n -> n <= 1 ? n : fib(n - 1) + fib(n - 2) }.memoize()

var start = System.currentTimeMillis()
assert fib(10) == 55
assert fib(50L) == 12586269025L
assert fib(100G) == 354224848179261915075G
println "Completed in ${System.currentTimeMillis() - start} ms"
```

See also @Memoized

Classes and records (Java)

```java
public class Point {
    private int x;
    private int y;

    public int x() {
        return this.x;
    }
    public int y() {
        return this.y;
    }

    public String toString() {
        /* ... */
    }
    public int hashCode() {
        /* ... */
    }
    public boolean equals(Object other) {
        /* ... */
    }
}
```

```java
public record Point(int x, int y) {
}
```

Classes and records (Groovy)

```
class Point {
    private int x
    private int y
    int x() { this.x }
    int y() { this.y }
    String toString() {
        /* ... */
    }
    int hashCode() {
        /* ... */
    }
    boolean equals(Object other) {
        /* ... */
    }
}
```

```
record Point(int x, int y) {
}
```

```
@RecordType
class Point {
    int x
    int y
}
```

Classes and records (Groovy)

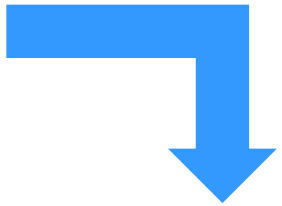
```
class Point {
    private int x
    private int y
    int x() { this.x }
    int y() { this.y }
    String toString() {
        /* ... */
    }
    int hashCode() {
        /* ... */
    }
    boolean equals(Object other) {
        /* ... */
    }
}
```

```
record Point(int x, int y) {
}
```

```
@RecordBase
@ToString
@EqualsAndHashCode
@RecordOptions
@TupleConstructor
@PropertyOptions
@KnownImmutable
class Point {
    int x
    int y
}
```

AST Transformations: @Immutable 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
}
```

Declarative Record Customization

```
record Agenda(List topics) { }
```

```
def a = new Agenda(topics: ['Sealed', 'Records'])
assert a.topics().size() == 2
assert a.toString() == 'Agenda[topics=[Sealed, Records]]'

a.topics().clear()
a.topics() << 'Switch Expressions'
assert a.topics().size() == 1
```

Declarative Record Customization

```
@ToString
@PropertyOptions(propertyHandler = ImmutablePropertyHandler)
record Agenda(List topics) { }
```

```
def a = new Agenda(topics: ['Sealed', 'Records'])
assert a.topics().size() == 2
shouldFail(UnsupportedOperationException) {
    a.topics().clear()
}
assert a.toString() == 'Agenda([Sealed, Records])'
```

Using records with AST Transforms: @Memoized @Builder

```
record Point(int x, int y, String color) {
    @Memoized
    String description() {
        "${color.toUpperCase()} point at ($x,$y)"
    }
}
```

```
record Developer(Integer id, String first, String last,
                 String email, List<String> skills) {
    @Builder
    Developer(Integer id, String full, String email, List<String> skills) {
        this(id, full.split(' ')[0], full.split(' ')[1], email, skills)
    }
}
```

Using records with AST Transforms: @Requires @Sortable

```
@Requires({ color && !color.blank })
record Point(int x, int y, String color) { }
```

```
@Sortable
record Point(int x, int y, String color) { }

var points = [
        new Point(0, 100, 'red'),
        new Point(10, 10, 'blue'),
        new Point(100, 0, 'green'),
]

println points.toSorted(Point.comparatorByX())
println points.toSorted(Point.comparatorByY())
println points.toSorted(Point.comparatorByColor())
```

Using records with AST Transforms: @Newify @OperatorRename

- Representing a quadratic equation: $ax^2 + bx + c$
  org.apache.commons:commons-numbers-core:1.1

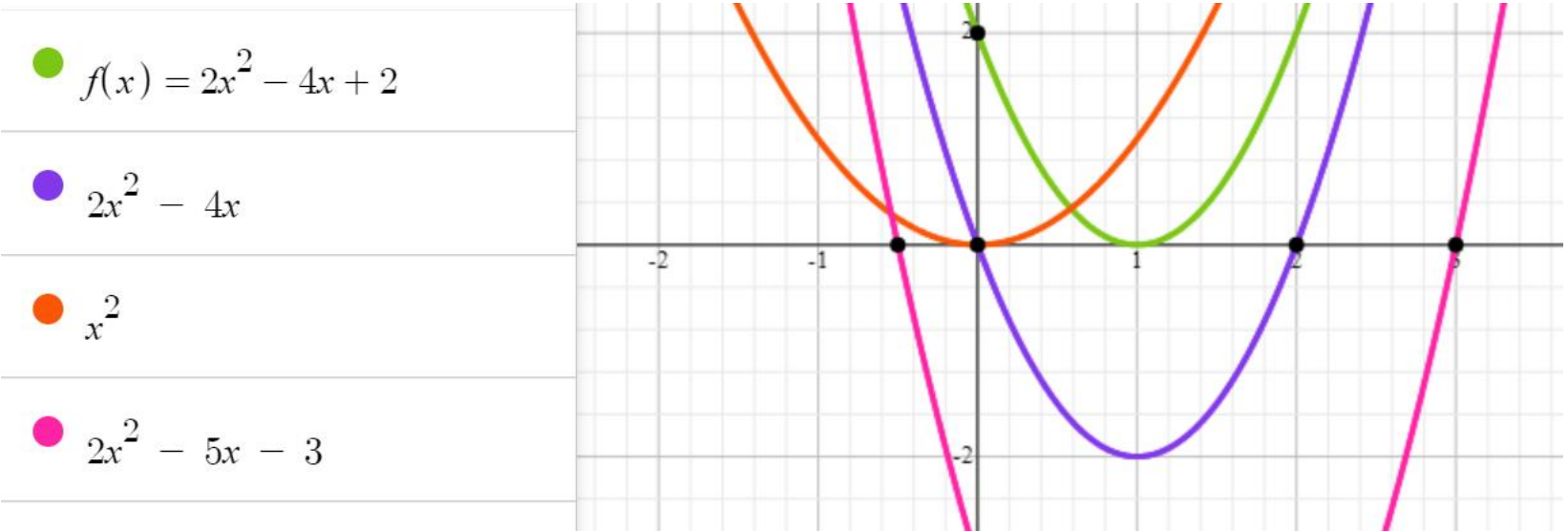
```
record Quadratic(double a, double b = 0, double c = 0) {
    @Newify(Complex)
    List<Complex> solve() {
        var discriminant = Complex(b * b - 4 * a * c)
        findRoots(Complex(-b), discriminant, Complex(2 * a))
    }

    @OperatorRename(div = 'divide', plus = 'add', minus = 'subtract')
    static List<Complex> findRoots(Complex negB, Complex discriminant, Complex twoA) {
        var sqrtDiscriminant = discriminant.sqrt()
        var root1 = (negB + sqrtDiscriminant) / twoA
        var root2 = (negB - sqrtDiscriminant) / twoA
        [root1, root2]
    }
}
```

Using records with other features: Named/Default params

```
assert [
        new Quadratic(2.0, -4.0, 2.0),
        new Quadratic(2.0, -4.0),
        new Quadratic(1.0),
        new Quadratic(a:2.0, b:-5.0, c:-3.0)
]*.solve()*.toSet()*.toString() == [
        '[(1.0, 0.0)]',
        '[(2.0, 0.0), (0.0, 0.0)]',
        '[(0.0, 0.0)]',
        '[(3.0, 0.0), (-0.5, 0.0)]'
]
```

Default parameters

Named parameters



Records: compared to Java

	Java Record	Groovy Emulated Record	Groovy Native Record
JDK version	16+	8+	16+
Serialization	Record spec	Traditional	Record spec
Recognized by	Java, Groovy	Groovy	Java, Groovy
Standard features • *accessors* • *tuple constructor* • *toString, equals, hashCode*	☑	☑	☑
Optional enhancements	☒	toMap, toList, size, getAt, components, copyWith, named-arg constructor, optional args	
Customisable via coding	☑	☑	☑
Customisable via AST transforms (declarative)	☒	☑	☑

Other improvements: switch expressions

```
class CustomIsCase {
    boolean isCase(subject) { subject > 20 }
}

assert switch(10) {
    case 0                    -> false
    case '11'                 -> false
    case null                 -> false
    case 0..9                 -> false
    case 1, 2                 -> false
    case [9, 11, 13]          -> false
    case Float                -> false
    case { it % 3 == 0 }      -> false
    case new CustomIsCase()   -> false
    case ~/\d\d/              -> true
    default                   -> false
}
```

Other improvements: switch expressions

- Duck & flow typing support styles which otherwise need special support when using static typing

```
String formatted = switch (o) {
    case Integer i when i > 10 -> String.format("a large Integer %d", i);
    case Integer i             -> String.format("a small Integer %d", i);
    case Long l                -> String.format("a Long %d", l);
    default                    -> o.toString();
};
```

JDK21 with preview features enabled

```
String formatted = switch (o) {
    case { o instanceof Integer && o > 10 } -> "a large Integer $o"
    case Integer                            -> "a small Integer $o"
    case Long                               -> "a Long $o"
    default                                 -> o.toString()
}
```

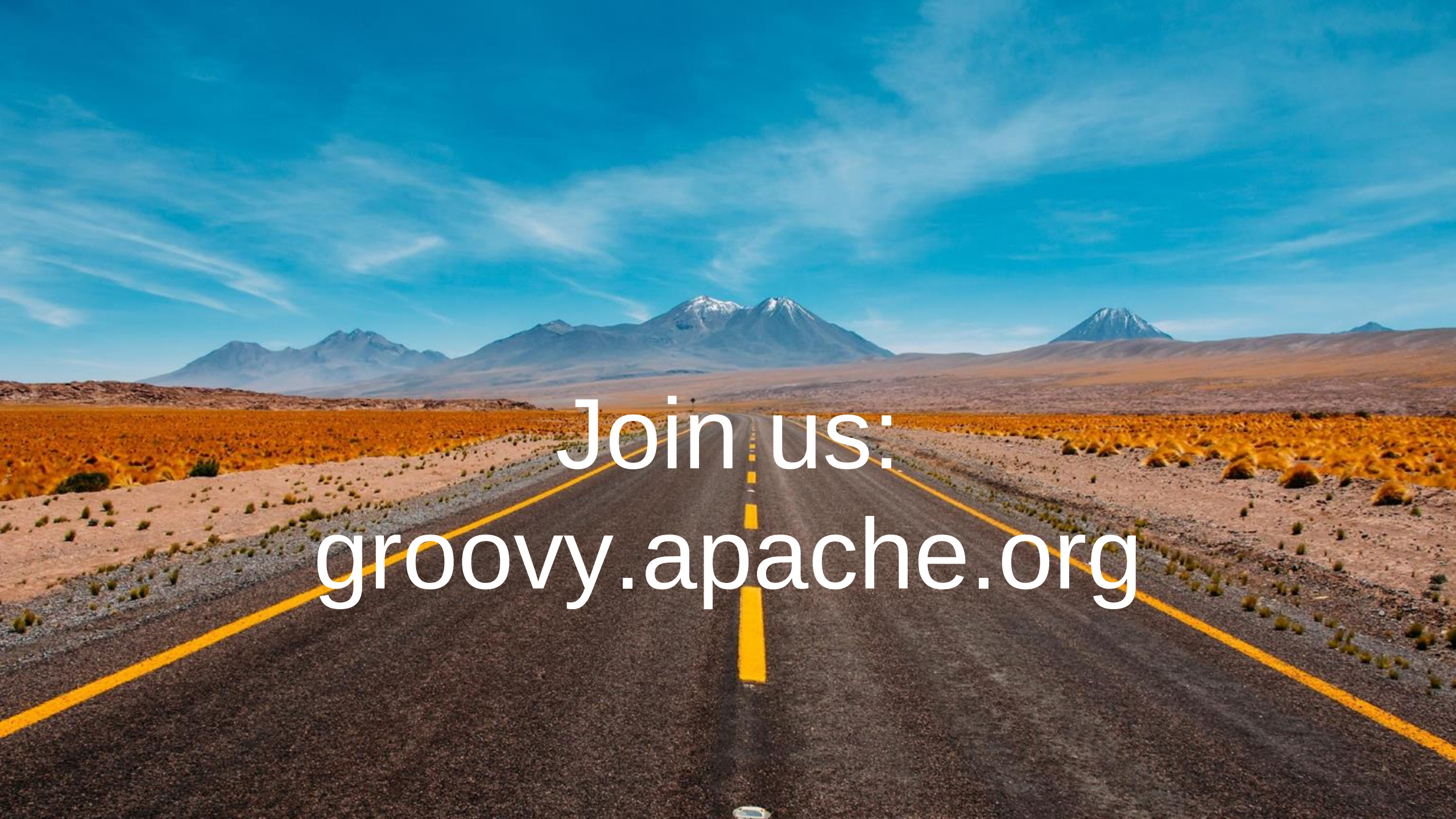
JDK8

Other improvements: switch expressions

	Java	Groovy
JDK versions	14+	8+
“ -> ” syntax (switch rule)	☑	☑
“ : … yield” syntax	☑	☑
Supported case selector expressions	Constant • Boolean, Number, String Enum constant	Constant • Boolean, Number, String, null Enum constant List expression Range expression Closure Regex Pattern Class expression
Enhanced switch (JDK21+)	Type pattern Guard/when Null	Class or Closure Closure already supported
Extensible via isCase	☒	☑

The Groovy future is looking good

- Groovy still has many features not yet available in Java
- Groovy adds much value to Java features
- Groovy has unparalleled extensibility so you can add features if there are some which are missing and you would like
- We shouldn’t be looking at Groovy replacing Java but rather use whichever make sense for the task at hand


Join us:
groovy.apache.org

Bonus material

- Better scripting
- Sealed types
- Sequenced collections

JEP 445 Scripting: Java

- Standard class:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- With JEP 445 Scripting (JDK 21 with preview enabled)

```
void main() {
    System.out.println("Hello, World!");
}
```

JEP 445 Scripting: Groovy earlier versions

- Standard class:

```
class HelloWorld {
    static main(args) {
        println 'Hello, World!'
    }
}
```

- Static main method only:

```
@CompileStatic
static main(args) {
    println 'Hello, World!'
}
```

- ***Usually only used if you want to annotate the main method***

- Script:

```
println 'Hello, World!'
```

JEP 445 Scripting: Groovy earlier versions

- Standard class:

```
class HelloWorld {
    static main(args) {
        println 'Hello, World!'
    }
}
```

- ***Return type promoted to*** `public void`
- ***Arguments promoted to*** `String[]`

- Static main method only:

```
@CompileStatic
static main(args) {
    println 'Hello, World!'
}
```

- ***Script class added***
- ***Has access to binding and context***

- Script:

```
println 'Hello, World!'
```

- ***Content added to*** `run` ***method***
- `public static void main` ***added creating instance and calling*** `run`
- ***Variable declarations are local variables in*** `run` ***method or promoted to fields if annotated with*** `@Field`

JEP 445 Scripting: Groovy 5

- "Instance main" method:

```
def main() {
    assert upper(foo) + lower(bar) == 'FOObar'
}

def upper(s) { s.toUpperCase() }

def lower = String::toLowerCase
def (foo, bar) = ['Foo', 'Bar']
```

- "Instance run" method:

```
@JsonIgnoreProperties(["binding"])
def run() {
    var mapper = new ObjectMapper()
    assert mapper.writeValueAsString(this) == '{"pets":["cat","dog"]}'
}

public pets = ['cat', 'dog']
```

- ***Now also supported:***
  - ***Instance*** `main` ***which are JEP 445 compatible***
  - ***Instance*** `run` ***which remains a script***
- `@Field` ***now only needed for standard scripts***

Scripting: compared to Java

	Earlier JDK versions	JDK 21 with preview enabled	Earlier Groovy versions	Groovy 5
Traditional Java Class	✓	✓	✓	✓
Traditional Groovy Script	✗	✗	✓ 1 2 5	✓ 1 2 5
"instance run" script	✗	✗	✗	✓ 1 2
"static main" script	✗	*JEP 445 Unnamed class & updated run protocol* 3	✓ 1 2 5	✓ 1
"instance main" script	✗		✗	✓ 3 4

1 Promoted to standard "public static void main"

2 Access to script binding & context

3 Use new run protocol

4 Runnable on JDK11+ from Groovy

5 Uses @Field

Other improvements: Sealed types

	Java Sealed Type	**Groovy Emulated Sealed Type**	**Groovy Native Sealed Type**
JDK version	17+	8+	17+
Recognized by	Java, Groovy	Groovy	Java, Groovy
“non-sealed” to reopen	Required	Optional	Optional

Other improvements: Sequenced collections (JDK 21)

First element	Java	Java with JEP-431	Groovy (JDK 8+)	Groovy with JEP-431

List	`list.get(0)`	`collection.getFirst()`	`aggregate[0]`
			`aggregate.first()`
				Adds:
				`collection.first`
				`collection.getFirst()`

Deque	`deque.getFirst()`			
Set	`set.iterator().next()` *or* `set.stream().findFirst().get()`			
SortedSet	`set.first()`			
array	`array[0]`	*unchanged*		*unchanged*

Last element	Java	Java with JEP-431	Groovy (JDK 8+)	Groovy with JEP-431

List	`list.get(list.size()-1)`	`collection.getLast()`	`aggregate[-1]`
			`aggregate.last()`
				Adds:
				`collection.last`
				`collection.getLast()`

Deque	`deque.getLast()`			
Set	*requires iterating through the set*			
SortedSet	`set.last()`			
array	`array[array.length-1]`	*unchanged*		*unchanged*

Other improvements: Gatherer-related functionality

	Groovy (Collections)	Java & Groovy (Streams)	Java & Groovy with JEP461
JDK versions	8+	8+	22 (*preview*)

Basic windowing	`take(n)`
	`drop(n)`
	ranges
		`limit(n)`
		`skip(n)`
		-
			-
			-
			-

Advanced windowing	`collate(n)`
	`collate(n, step)`
	`collate(n, step, truncate)`
	`chop(n1, n2, …)`
		-
		-
		-
		-
			`windowFixed(n)`
			`windowSliding(n)`
			custom gatherer
			custom gatherer

Inject/fold	`inject` (*homogeneous types*)
	`inject` (*heterogenous types*)
	`inits()`
	`tails()`
		`reduce()`
		-
		-
		-
			-
			`fold()`
			custom gatherer
			custom gatherer

Cumulative sum	*`inits()` and `sum()`*	*Non-stream:* `Arrays.parallelPrefix()`	`scan()`

See: https://groovy.apache.org/blog/groovy-gatherers