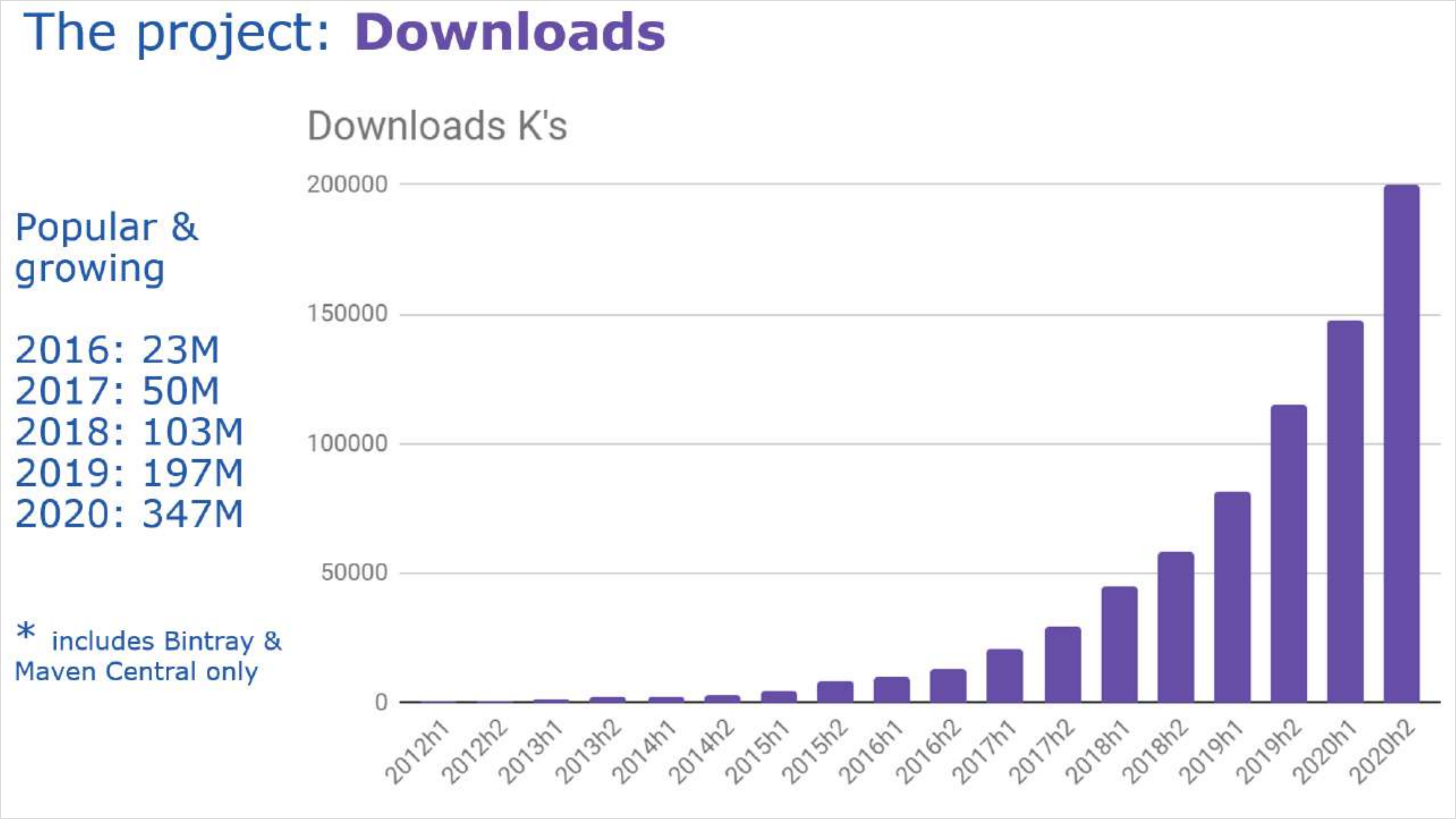
An Insider's Guide to Groovy 4

Presented by
Dr Paul King

objectcomputing.com

Groovy – Downloads increasing

- > 1B downloads and growing



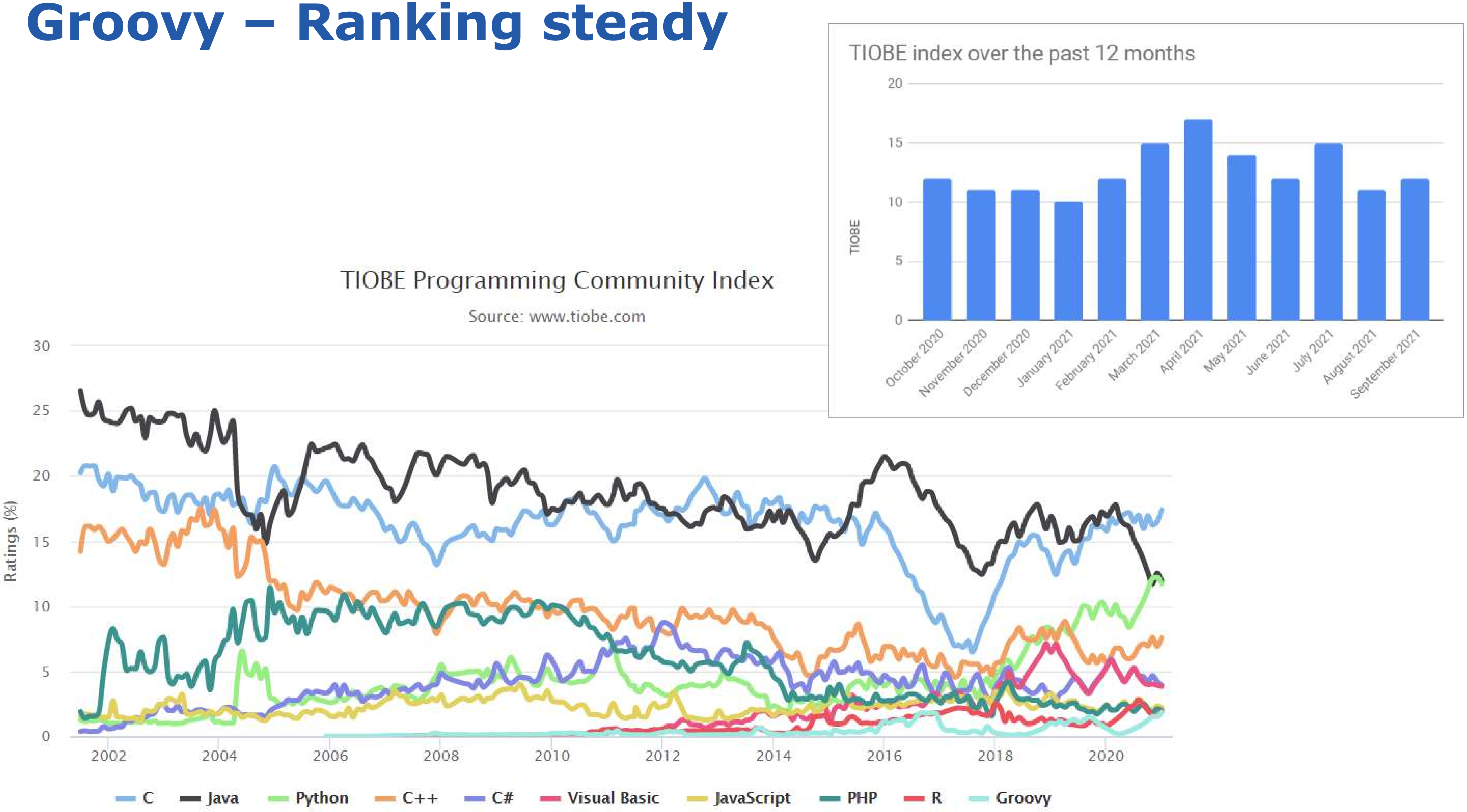
The project: **Downloads**

Popular & growing

2016: 23M
2017: 50M
2018: 103M
2019: 197M
2020: 347M

* includes Bintray & Maven Central only

Groovy – Ranking steady



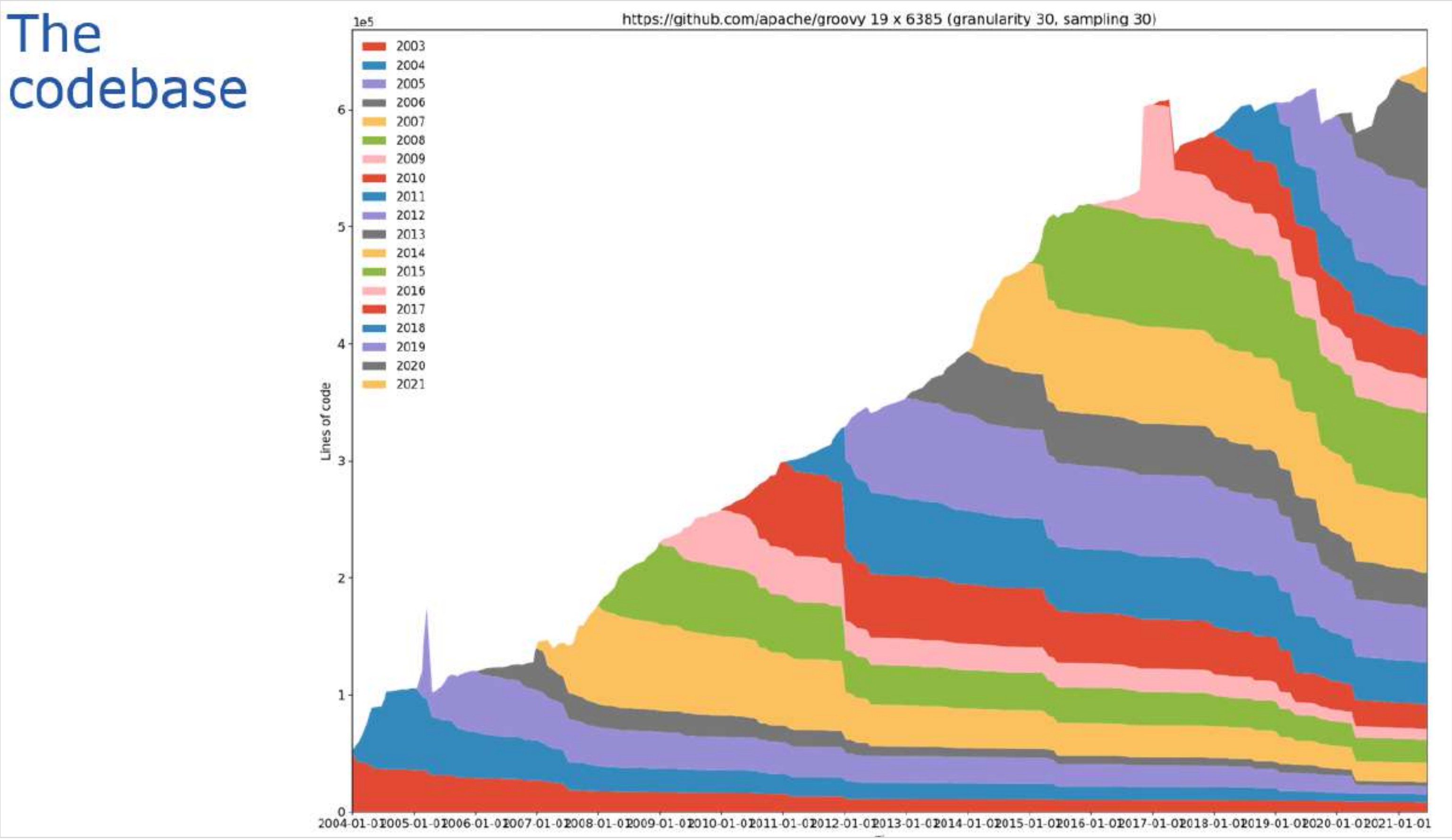
TIOBE index over the past 12 months

TIOBE Programming Community Index

Source: www.tiobe.com

Groovy – Activity steady

- > 600K lines of source code
- > 19K commits
- > 8K issues & enhancements resolved
- > 500 contributors
- > 200 releases



Groovy 4 - Summary

Consolidation & Structuring

- Maven coordinates
- Module changes
- Indy only, Parrot only
- ~33% smaller zip
- ~10% smaller core jar

Language Features

- Switch expressions
- Sealed types
- Improved type annotations
- Language integrated query

Libraries/Tooling

- Built-in type checkers
- Built-in macro methods
- TOML builder/slurper
- JavaShell
- Improved ranges

AST transforms

- @POJO
- @RecordType
- Groovy Contracts

GDK enhancements

Groovy 4 - Summary

Consolidation & Structuring

- Maven coordinates
- Module changes
- Indy only, Parrot only
- ~33% smaller zip
- ~10% smaller core jar

Language Features

- Switch expressions
- Sealed types
- Improved type annotations
- Language integrated query

Libraries/Tooling

- Built-in type checkers
- Built-in macro methods
- TOML builder/slurper
- JavaShell
- Improved ranges

AST transforms

- @POJO
- @RecordType
- Groovy Contracts

GDK enhancements

Important naming/structuring changes

Maven coordinate change

~~org.codehaus.groovy~~ → org.apache.groovy

Important naming/structuring changes

Maven coordinate change

~~org.codehaus.groovy~~ → org.apache.groovy

Note: Doesn’t imply all internal package names have been changed.

Module changes

Removed modules

~~groovy-bsf~~ ~~groovy-jaxb~~

New optional modules

groovy-contracts

groovy-ginq

groovy-macro-library

groovy-toml

groovy-typecheckers

Module changes for groovy-all

groovy-testng: included in all → *optional*

groovy-yaml: optional → *included in all*

Module changes

Split packaging legacy package removal

groovy-xml:

2.5	3.0	4.0
groovy.util.XmlParser	~~groovy.util.XmlParser~~	
groovy.util.XmlSlurper	~~groovy.util.XmlSlurper~~	
	groovy.xml.XmlParser	groovy.xml.XmlParser
	groovy.xml.XmlSlurper	groovy.xml.XmlSlurper

Also: groovy-ant, groovy-swing, groovy-test, …

More details:

https://groovy-lang.org/releasenotes/groovy-3.0.html#Groovy3.0releasenotes-Splitpackages

Legacy consolidation

Old parser removal

~~Antlr 2~~ Antlr4

Classic bytecode generation removal

~~Classic~~ Indy

Groovy 4 - Summary

Consolidation & Structuring

- Maven coordinates
- Module changes
- Indy only, Parrot only
- ~33% smaller zip
- ~10% smaller core jar

Language Features

- Switch expressions
- Sealed types
- Improved type annotations
- Language integrated query

Libraries/Tooling

- Built-in type checkers
- Built-in macro methods
- TOML builder/slurper
- JavaShell
- Improved ranges

AST transforms

- @POJO
- @RecordType
- Groovy Contracts

GDK enhancements

GINQ

Incubating

```
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

from p in persons
groupby p.gender
having p.gender == 'Male'
select p.gender, max(p.age)

from p in persons
orderby p.age in desc, p.name
select p.name

from n in numbers
where n > 0 && n <= 3
select n * 2

from n1 in nums1
innerjoin n2 in nums2 on n1 == n2
select n1 + 1, n2
```

Sealed Type Motivation

- Inheritance is a powerful abstraction for building systems

```
class Shape { … }
final class Square extends Shape { … }
final class Circle extends Shape { … }
```

Sealed Type Motivation

- Inheritance is a powerful abstraction for building systems
- There are scenarios where limiting inheritance has benefits
  - *Less defensive programming in parent classes*
  - *To support additional compiler checks, e.g. pattern matching/casts*
- Traditional mechanisms for limiting inheritance are crude
  - *Using* `final` *stops all inheritance (not applicable to interfaces)*
  - *Package-private parent classes don’t provide an accessible parent abstraction*

Sealed Type Motivation

- Inheritance is a powerful abstraction for building systems
- There are scenarios where limiting inheritance has benefits
  - *Less defensive programming in parent classes*
  - *To support additional compiler checks, e.g. pattern matching/casts*
- Traditional mechanisms for limiting inheritance are crude
  - *Using* `final` *stops all inheritance (not applicable to interfaces)*
  - *Package-private parent classes don't provide an accessible parent abstraction*
- Sealed type
  - *Provides a fixed set of children rather than all or nothing*
  - *Decouples accessibility from extendibility*
  - *Easier to add new methods, harder to add new types*
- Unsealed type
  - *Easy to add new types, harder to add new methods*

Sealed Types

```
@Sealed(permittedSubclasses=[Diamond,Circle]) class Shape { }
final class Diamond extends Shape { }
final class Circle extends Shape { }
```

- *Class or abstract class*
- *Annotation style*

Sealed Types

```
@Sealed(permittedSubclasses=[Diamond,Circle]) class Shape { }
final class Diamond extends Shape { }
final class Circle extends Shape { }
```

```
sealed trait Triangle permits Equilateral, Isosceles { }
final class Equilateral implements Triangle { }
final class Isosceles implements Triangle { }
```

- ***Trait***
- ***Keyword style***

Sealed Types

```
@Sealed(permittedSubclasses=[Diamond,Circle]) class Shape { }
final class Diamond extends Shape { }
final class Circle extends Shape { }
```

```
sealed trait Triangle permits Equilateral, Isosceles { }
final class Equilateral implements Triangle { }
final class Isosceles implements Triangle { }
```

```
sealed interface Polygon { }
final class Square implements Polygon { }
final class Rectangle implements Polygon { }
```

- ***Interface***
- ***Keyword style***
- ***Inferred subclasses***

Sealed Types – Good for ADTs

```
@Sealed interface Tree<T> {}

@Singleton final class Empty implements Tree {
    String toString() { 'Empty' }
}

@Canonical final class Node<T> implements Tree<T> {
    T value
    Tree<T> left, right
}

Tree<Integer> tree = new Node<>(42,
    new Node<>(0, Empty.instance, Empty.instance), Empty.instance)

assert tree.toString() == 'Node(42, Node(0, Empty, Empty), Empty)'
```

Sealed Types – Hybrid hierarchies

```
sealed class Shape permits Circle, Polygon, Rectangle { }

final class Circle extends Shape { }

non-sealed class Polygon extends Shape { }
final class Pentagon extends Polygon { }

sealed class Rectangle extends Shape permits Square { }
final class Square extends Rectangle { }
```

Sealed Types – Hybrid hierarchies

```
sealed class Shape permits Circle, Polygon, Rectangle { }

final class Circle extends Shape { }

non-sealed class Polygon extends Shape { }
final class Pentagon extends Polygon { }

sealed class Rectangle extends Shape permits Square { }
final class Square extends Rectangle { }
```

- ***Groovy follows Scala style of non-sealed being optional***
- ***We envisage a future CodeNarc rule which could enforce the Java style***

Switch expressions

```
def a = 9
def result = switch(a) {
    case 6, 8 -> 'b'
    case 9 -> 'c'
    default -> 'z'
}
assert 'c' == result
```

Switch expressions

```
enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday }
```

```
import static Day.*

def isWeekend(Day d) {
    switch(d) {
        case Monday..Friday -> false
        case [Sunday, Saturday] -> true
    }
}

assert [Sunday, Monday, Friday].collect{ isWeekend(it) }
       == [true, false, false]
```

Switch expressions

```
enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday }
```

```
import static Day.*

def isWeekend(Day d) {
    return switch(d) {
        case Monday..Friday: yield false
        case [Sunday, Saturday]: yield true
    }
}

assert [Sunday, Monday, Friday].collect{ isWeekend(it) }
       == [true, false, false]
```

Type Annotations

```
@Grab('net.jqwik:jqwik:1.5.5')
import net.jqwik.api.*
import net.jqwik.api.constraints.*

class PropertyBasedTests {
  @Property
  def uniqueInList(@ForAll @Size(5) @UniqueElements List<@IntRange(min = 0, max = 10) Integer> aList) {
    assert aList.size() == aList.toSet().size()
    assert aList.every{ anInt -> anInt >= 0 && anInt <= 10 }
  }
}
```

- Existing support
- Now supported

Type Annotations

```
@Grab('org.hibernate.validator:hibernate-validator:7.0.1.Final')
@Grab('org.hibernate.validator:hibernate-validator-cdi:7.0.1.Final')
@Grab('org.glassfish:jakarta.el:4.0.0')
import jakarta.validation.constraints.*
import jakarta.validation.*
import groovy.transform.*

@Canonical
class Car {
    @NotNull @Size(min = 2, max = 14) String make
    @Min(1L) int seats
    List<@NotBlank String> owners
}

def validator = Validation.buildDefaultValidatorFactory().validator

def violations = validator.validate(new Car(make: 'T', seats: 1))
assert violations*.message == ['size must be between 2 and 14']

violations = validator.validate(new Car(make: 'Tesla', owners: ['']))
assert violations*.message.toSet() == ['must be greater than or equal to 1', 'must not be blank'] as Set

violations = validator.validate(new Car(make: 'Tesla', owners: ['Elon'], seats: 2))
assert !violations
```

Groovy 4 - Summary

Consolidation & Structuring

- Maven coordinates
- Module changes
- Indy only, Parrot only
- ~33% smaller zip
- ~10% smaller core jar

Language Features

- Switch expressions
- Sealed types
- Improved type annotations
- Language integrated query

Libraries/Tooling

- Built-in type checkers
- Built-in macro methods
- TOML builder/slurper
- JavaShell
- Improved ranges

AST transforms

- @POJO
- @RecordType
- Groovy Contracts

GDK enhancements

Built-in type checkers: regex checker

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

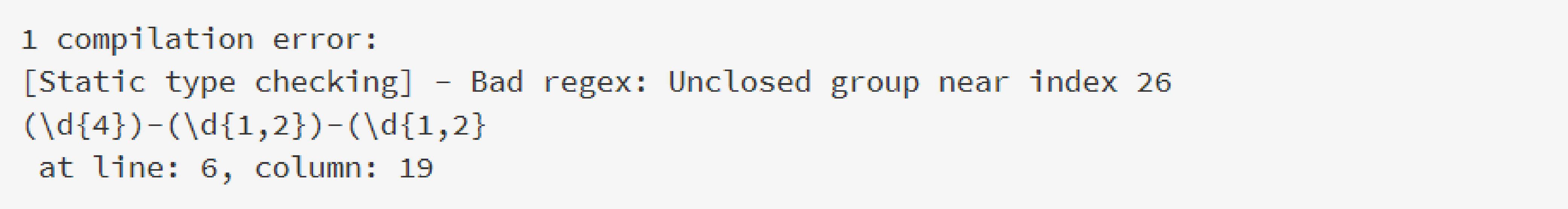
Built-in type checkers: regex checker

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

Built-in type checkers: regex 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
```

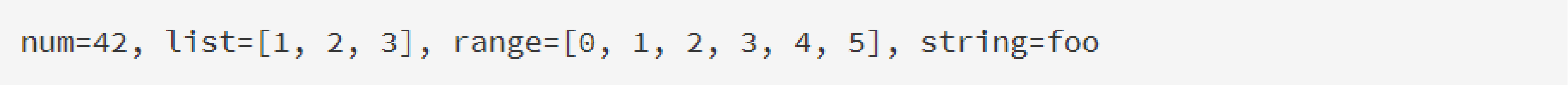
Built-in type checkers: regex checker

```
~/\w{3/                   // missing closing repetition quantifier brace
~"(.)o(.*"                // missing closing group bracket
Pattern.compile(/?/)      // dangling meta character '?' (Java longhand)

'foobar'  =~ /f[o]{2/          // missing closing repetition quantifier brace
'foobar' ==~ /(foo/            // missing closing group bracket
Pattern.matches(/?/, 'foo')    // dangling meta character '?' (Java longhand)

def m = 'foobar' =~ /(...)(...)/
assert m[0][1] == 'foo'             // okay
assert m[0][3]                      // type error: only two groups in regex

Pattern p = Pattern.compile('(...)(...)')
Matcher m = p.matcher('foobar')
assert m.find()
assert m.group(1) == 'foo'          // okay
assert m.group(3)                   // type error: only two groups in regex
```

Built-in macro methods

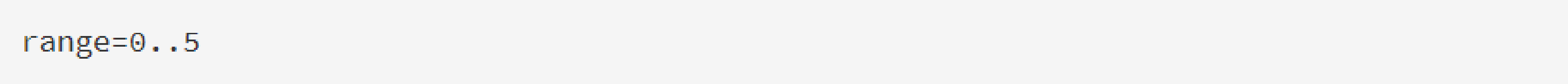
```
def num = 42
def list = [1 ,2, 3]
def range = 0..5
def string = 'foo'
```

```
println NV(num, list, range, string)
```



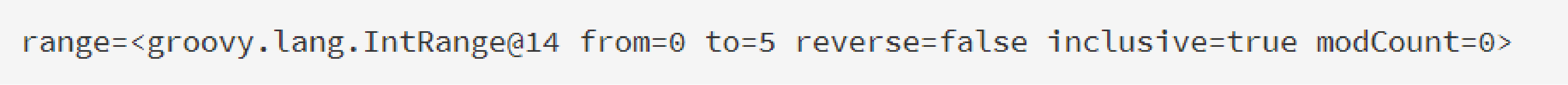
```
num=42, list=[1, 2, 3], range=[0, 1, 2, 3, 4, 5], string=foo
```

```
println NVI(range)
```



```
range=0..5
```

```
println NVD(range)
```



```
range=<groovy.lang.IntRange@14 from=0 to=5 reverse=false inclusive=true modCount=0>
```

TOML Builder (Incubating)

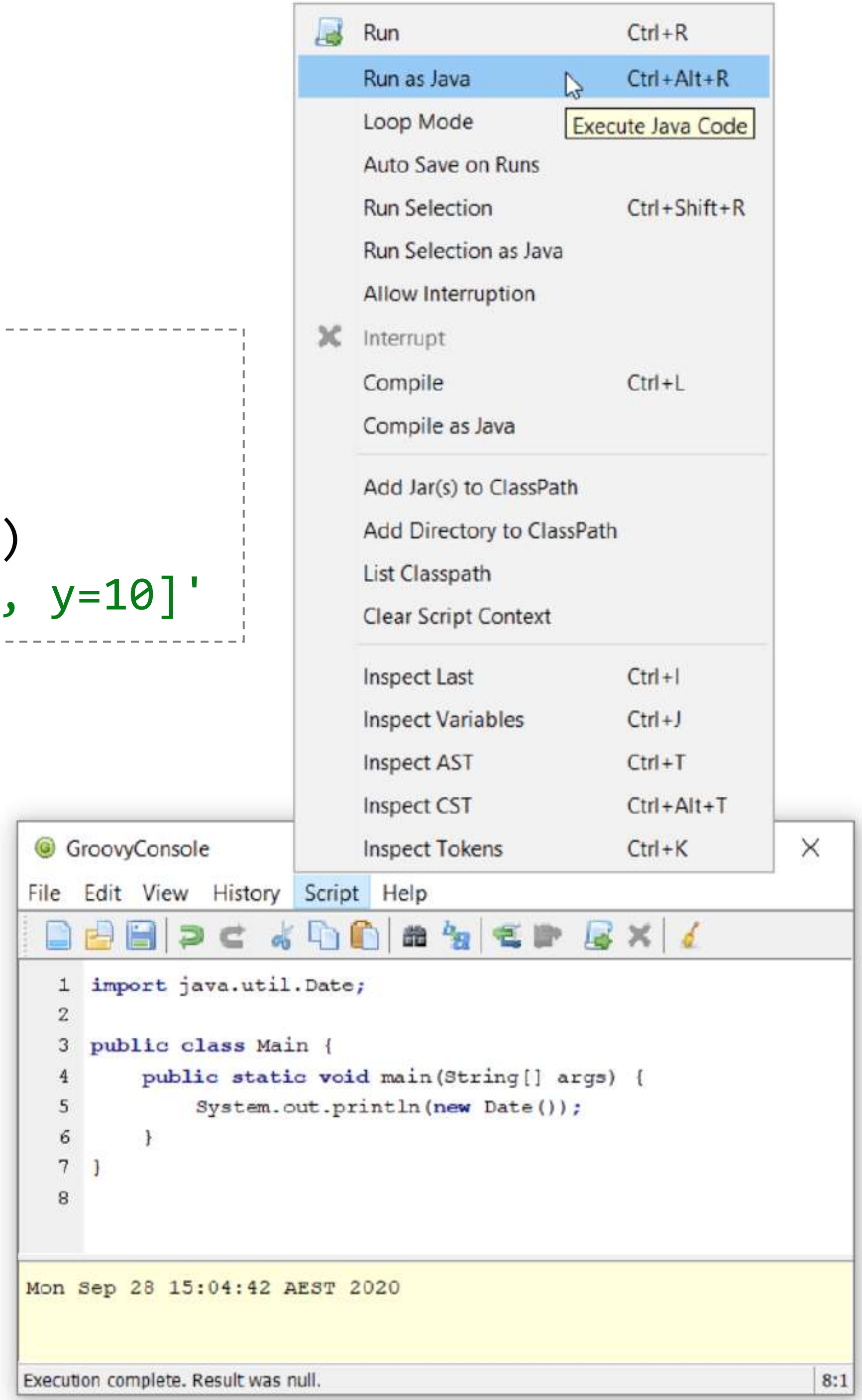
```
def builder = new TomlBuilder()
builder.records {
    car {
        name 'HSV Maloo'
        make 'Holden'
        year 2006
        country 'Australia'
        homepage new URL('http://example.org')
        record {
            type 'speed'
            description 'production pickup truck with speed of 271kph'
        }
    }
}
```

TOML Slurper (Incubating)

```
def ts = new TomlSlurper()
def toml = ts.parseText(builder.toString())

assert 'HSV Maloo' == toml.records.car.name
assert 'Holden' == toml.records.car.make
assert 2006 == toml.records.car.year
assert 'Australia' == toml.records.car.country
assert 'http://example.org' == toml.records.car.homepage
assert 'speed' == toml.records.car.record.type
assert 'production pickup truck with speed of 271kph' == toml.records.car.record.description
```

JavaShell

```
import org.apache.groovy.util.JavaShell
def opts = ['--enable-preview', '--release', '14']
def src = 'record Coord(int x, int y) {}'
Class coordClass = new JavaShell().compile('Coord', opts, src)
assert coordClass.newInstance(5, 10).toString() == 'Coord[x=5, y=10]'
```



```
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        System.out.println(new Date());
    }
}

```

```
Mon Sep 28 15:04:42 AEST 2020
```

Execution complete. Result was null.

Improved Ranges

```
def range = 1..5
assert range == [1, 2, 3, 4, 5]

range = 1..<5
assert range == [1, 2, 3, 4]

range = 1<..5
assert range == [2, 3, 4, 5]

range = 1<..<5
assert range == [2, 3, 4]
```

- Existing support
- Now supported

Groovy 4 - Summary

Consolidation & Structuring

- Maven coordinates
- Module changes
- Indy only, Parrot only
- ~33% smaller zip
- ~10% smaller core jar

Language Features

- Switch expressions
- Sealed types
- Improved type annotations
- Language integrated query

Libraries/Tooling

- Built-in type checkers
- Built-in macro methods
- TOML builder/slurper
- JavaShell
- Improved ranges

AST transforms

- @POJO
- @RecordType
- Groovy Contracts

GDK enhancements

AST Transformations

```
class Book {

    List<String> authors

    String title

    Date publicationDate
}
```

AST Transformations

```
class Book {

    List<String> authors

    String title

    Date publicationDate
}
```

```
public class Book implements GroovyObject {

    private java.util.List<String> authors
    private java.lang.String title
    private java.util.Date publicationDate

    public java.util.List<String> getAuthors() { ... }

    public void setAuthors(java.util.List<String> value) { ... }

    public java.lang.String getTitle() { ... }

    public void setTitle(java.lang.String value) { ... }

    public java.util.Date getPublicationDate() { ... }

    public void setPublicationDate(java.util.Date value) { ... }

}
```

AST Transformations

```
@ToString
class Book {

    List<String> authors

    String title

    Date publicationDate
}
```

AST Transformations

```
@ToString
class Book {

    List<String> authors

    String title

    Date publicationDate
}
```

```
public class Book implements GroovyObject {

    private java.util.List<String> authors
    private java.lang.String title
    private java.util.Date publicationDate

    public java.util.List<String> getAuthors() { ... }

    public void setAuthors(java.util.List<String> value) { ... }

    public java.lang.String getTitle() { ... }

    public void setTitle(java.lang.String value) { ... }

    public java.util.Date getPublicationDate() { ... }

    public void setPublicationDate(java.util.Date value) { ... }

    public java.lang.String toString() {
        /* build toString based on properties */
    }

}
```

AST Transformations

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

    String title

    Date publicationDate
}
```

AST Transformations

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

    String title

    Date publicationDate
}
```

```java
// 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<String>) ((ArrayList<?>) aut
                this.authors = (List<String>) (authorsCopy instanceof Sorted
                        : authorsCopy instanceof SortedMap ? DefaultGroovyM
                        : authorsCopy instanceof Set ? DefaultGroovyMethods.
                        : authorsCopy instanceof Map ? DefaultGroovyMethods
                        : authorsCopy instanceof List ? DefaultGroovyMethods.
                        : DefaultGroovyMethods.asImmutable(authorsCopy));
            } else {
                this.authors = (List<String>) (authors instanceof SortedSet ?
                        : authors instanceof SortedMap ? DefaultGroovyMetho
                        : authors instanceof Set ? DefaultGroovyMethods.asIm
                        : authors instanceof Map ? DefaultGroovyMethods.asIr
                        : authors instanceof List ? DefaultGroovyMethods.asIm
                        : DefaultGroovyMethods.asImmutable(authors));
            }
        }
        this.title = title;
        if (publicationDate == null) {
            this.publicationDate = null;
        } else {
            this.publicationDate = (Date) publicationDate.clone();
        }
    }

    public Book(Map args) {
        if ( args == null) {
            args = new HashMap();
        }
        ImmutableASTTransformation.checkPropNames(this, args);
        if (args.containsKey("authors")) {
            if ( args.get("authors") == null) {
                this .authors = null;
            } else {
                if (args.get("authors") instanceof Cloneable) {
                    List<String> authorsCopy = (List<String>) ((ArrayList<?>) a
                    this.authors = (List<String>) (authorsCopy instanceof Sorte
                            : authorsCopy instanceof SortedMap ? DefaultGroovy
                            : authorsCopy instanceof Set ? DefaultGroovyMethod
                            : authorsCopy instanceof Map ? DefaultGroovyMetho
                            : authorsCopy instanceof List ? DefaultGroovyMethod
                            : DefaultGroovyMethods.asImmutable(authorsCopy)
                } else {
                    List<String> authors = (List<String>) args.get("authors");
                    this.authors = (List<String>) (authors instanceof SortedSet
                            : authors instanceof SortedMap ? DefaultGroovyMethods.asImmutable(authors)
                            : authors instanceof Set ? DefaultGroovyMethods.asImmutable(authors)
                            : authors instanceof Map ? DefaultGroovyMethods.asImmutable(authors)
                            : authors instanceof List ? DefaultGroovyMethods.asImmutable(authors)
                            : DefaultGroovyMethods.asImmutable(authors));
                }
            }
        } else {
            this .authors = null;
        }
        if (args.containsKey("title")) {this .title = (String) args.get("title"); } else { this .title = null;}
        if (args.containsKey("publicationDate")) {
            if (args.get("publicationDate") == null) {
                this.publicationDate = null;
            } else {
                this.publicationDate = (Date) ((Date) args.get("publicationDate")).clone();
            }
        } else {this.publicationDate = null; }
    }
...
```

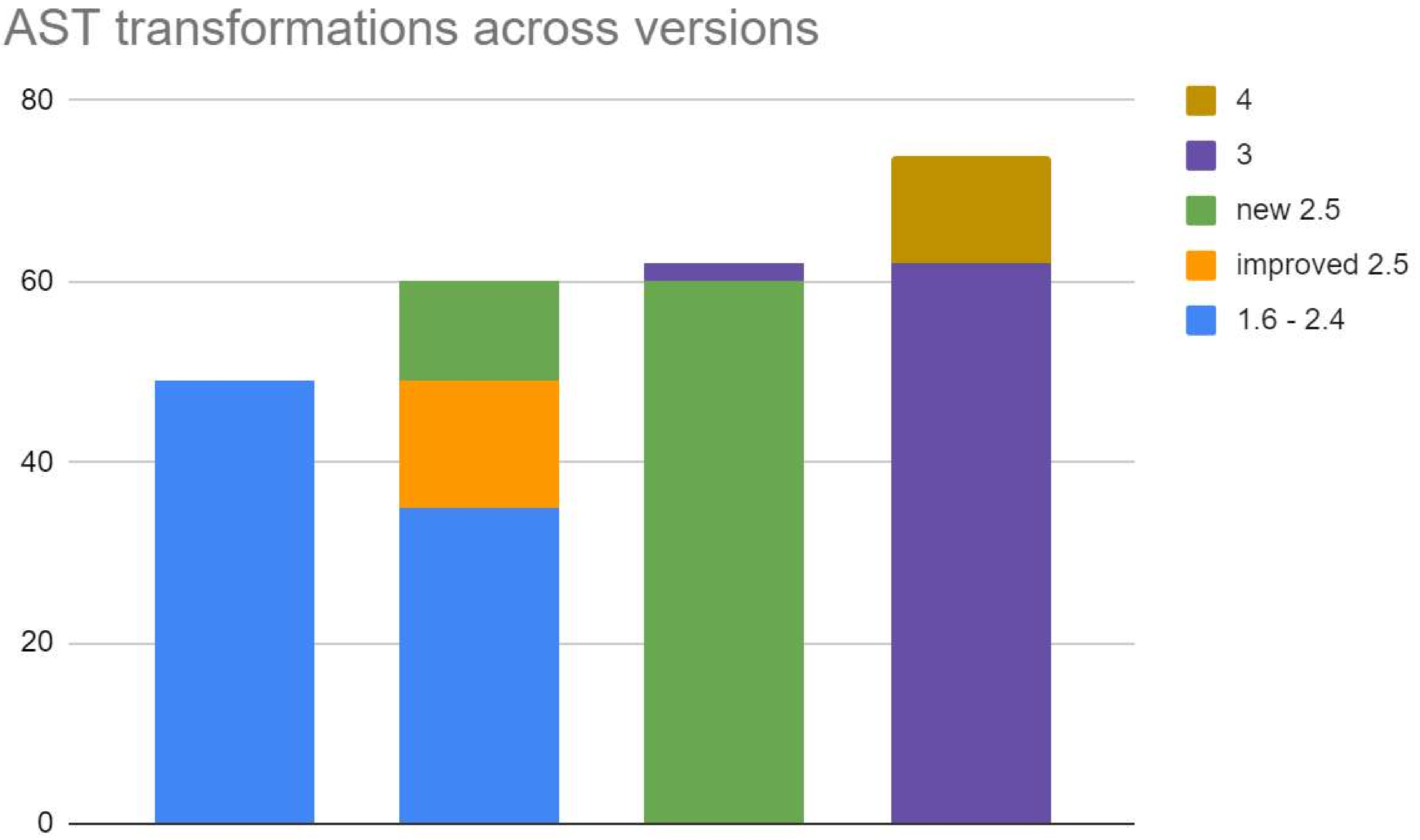
```java
    public Book() {
        this (new HashMap());
    }

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

```java
                _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;
    }

...
```

```java
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();
}
...
```

```java
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 2.4, Groovy 2.5, Groovy 3.0, Groovy 4.0



AST Transformations: Groovy 2.4, Groovy 2.5, Groovy 3.0, Groovy 4.0

(Improved in 2.5)

@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

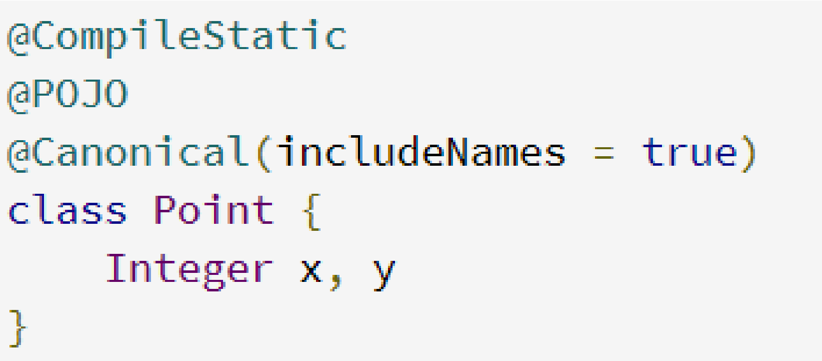
@ListenerList
@~~Mixin~~

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

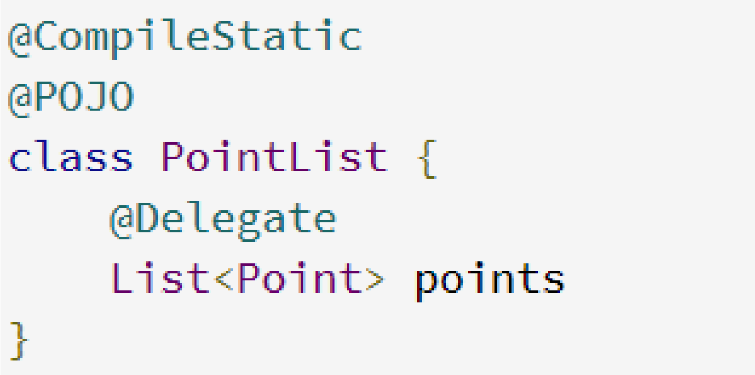
@AutoFinal
@AutoImplement
@ImmutableBase
@ImmutableOptions
@MapConstructor
@NamedDelegate
@NamedParam
@NamedParams
@NamedVariant
@PropertyOptions
@VisibilityOptions
@GroovyDoc
@NullCheck

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

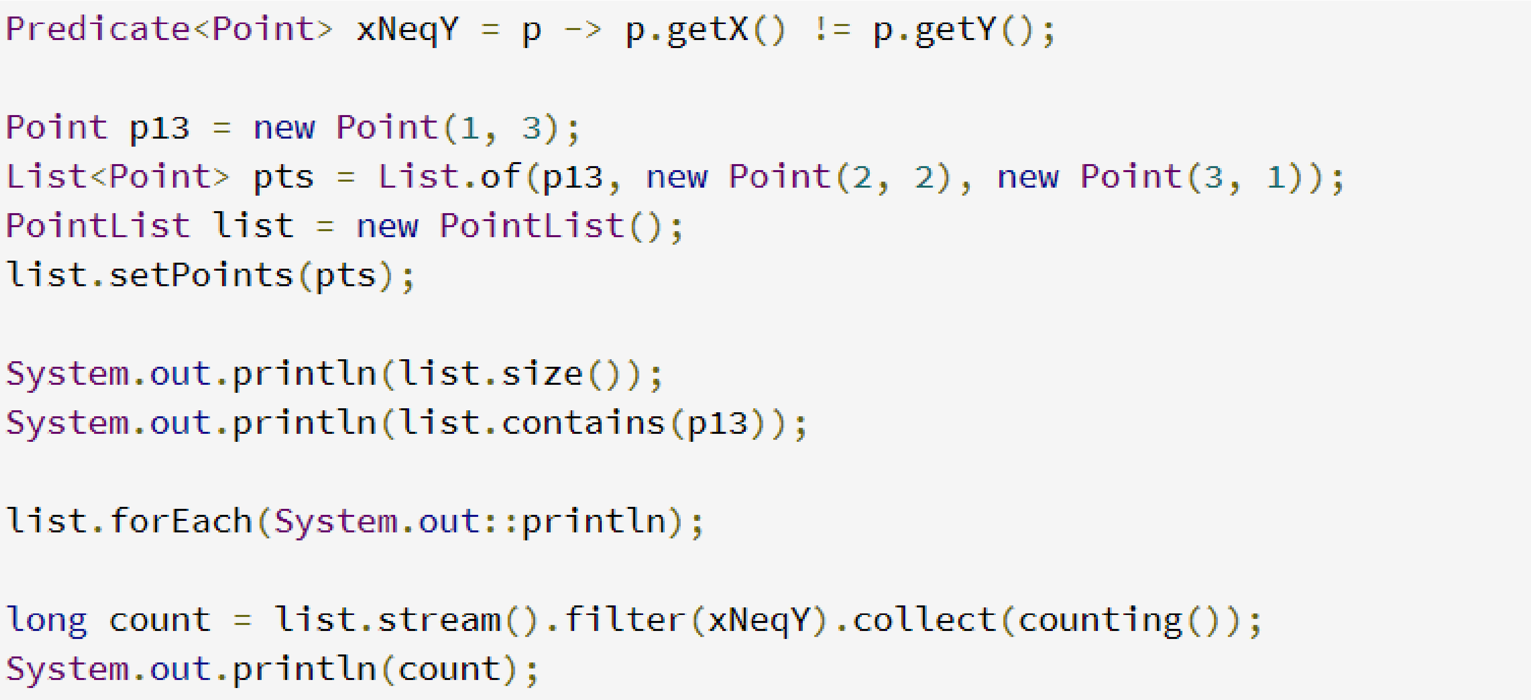
@POJO (incubating)



```
@CompileStatic
@POJO
@Canonical(includeNames = true)
class Point {
    Integer x, y
}
```



```
@CompileStatic
@POJO
class PointList {
    @Delegate
    List<Point> points
}
```



```
Predicate<Point> xNeqY = p -> p.getX() != p.getY();

Point p13 = new Point(1, 3);
List<Point> pts = List.of(p13, new Point(2, 2), new Point(3, 1));
PointList list = new PointList();
list.setPoints(pts);

System.out.println(list.size());
System.out.println(list.contains(p13));

list.forEach(System.out::println);

long count = list.stream().filter(xNeqY).collect(counting());
System.out.println(count);
```

```
3
true
Point(x:1, y:3)
Point(x:2, y:2)
Point(x:3, y:1)
2
```

Groovy 2.5: 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
}
```

Groovy 4.0: AST Transforms: @RecordType meta-annotation

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

```groovy
@RecordBase
@ToString(cache = true, includeNames = true)
@EqualsAndHashCode(cache = true, useCanEqual = false)
@ImmutableOptions
@PropertyOptions(propertyHandler = ImmutablePropertyHandler)
@TupleConstructor(defaults = false)
@MapConstructor
@KnownImmutable
@POJO
class Point {
    int x, y
}
```

@RecordType

```
@RecordType
class Cyclist {
    String firstName
    String lastName
}

def richie = new Cyclist('Richie', 'Porte')
```

Produces a class that:

- *is implicitly final*
- *has a private final field* firstName *with an accessor method firstName();* *ditto for* lastName
- *has a default* Cyclist(String, String) *constructor*
- *has a default* serialVersionUID *of* 0L
- *has implicit* toString(), equals() *and* hashCode() *methods*

```
record Cyclist(String firstName, String lastName) { }    // possible future syntax
```

groovy-contracts module

Design-by-contract

```
import groovy.contracts.*

@Invariant({ speed() >= 0 })
class Rocket {
    int speed = 0
    boolean started = true

    @Requires({ isStarted() })
    @Ensures({ old.speed < speed })
    def accelerate(inc) { speed += inc }

    def isStarted() { started }

    def speed() { speed }
}

def r = new Rocket()
r.accelerate(5)
```

Groovy 4 - Summary

Consolidation & Structuring

- Maven coordinates
- Module changes
- Indy only, Parrot only
- ~33% smaller zip
- ~10% smaller core jar

Language Features

- Switch expressions
- Sealed types
- Improved type annotations
- Language integrated query

Libraries/Tooling

- Built-in type checkers
- Built-in macro methods
- TOML builder/slurper
- JavaShell
- Improved ranges

AST transforms

- @POJO
- @RecordType
- Groovy Contracts

GDK enhancements

GDK Enhancements

```
assert (Stream.of(1) + Stream.of(2)).toList() == [1,2]
```

```
println Runtime.runtime.pid
```

Still being explored for future Groovy versions

- Additional `switch` destructuring/pattern matching
- `instanceof` “pattern matching”
- Smarter type checking: non-null, pure
- Module definitions in Groovy
- AST transform priority
- Syntactic sugar wrapper for JDK11 HttpClient
- Record syntactic sugar and native records