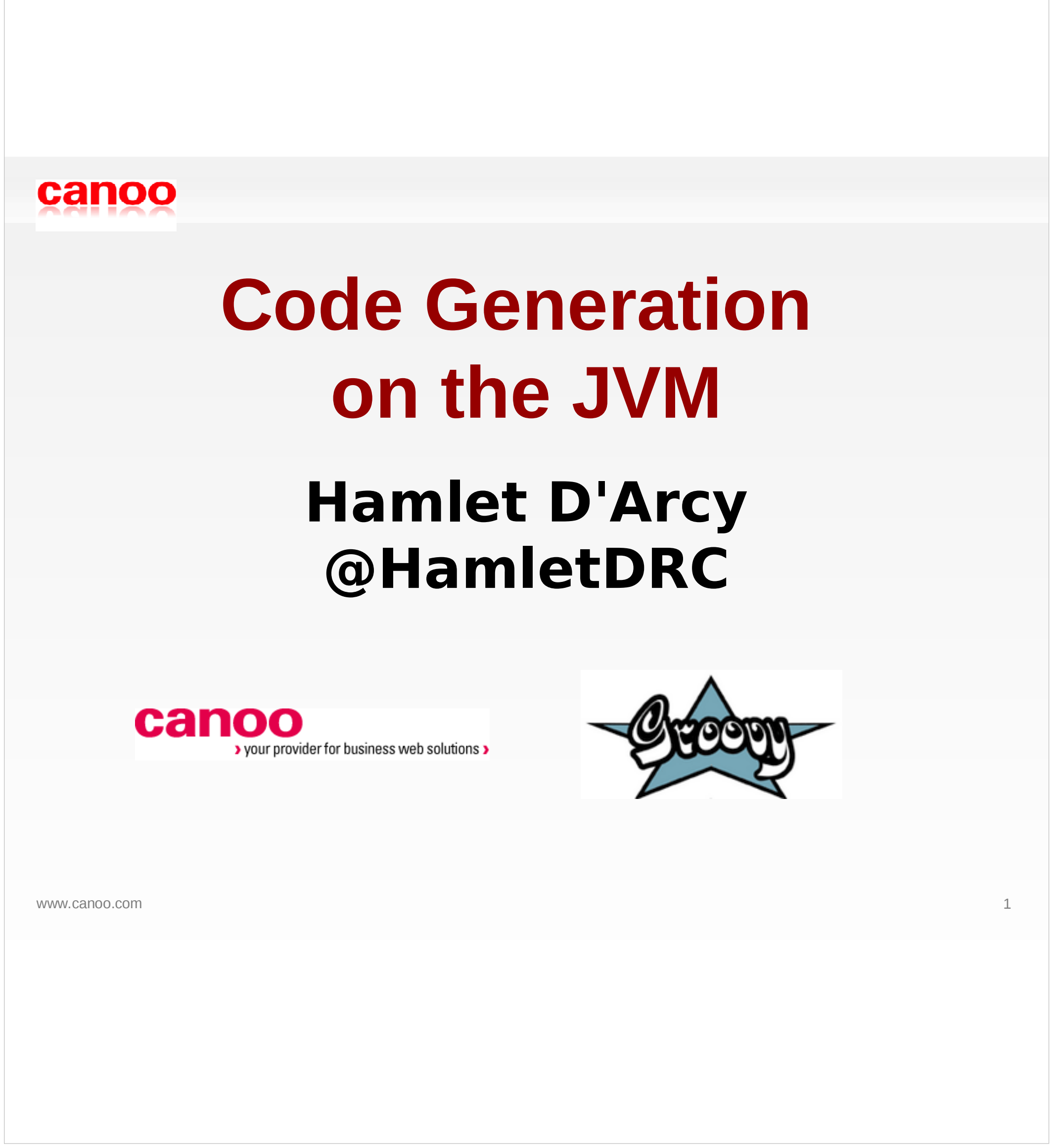


canoo

Code Generation on the JVM

Hamlet D'Arcy
@HamletDRC

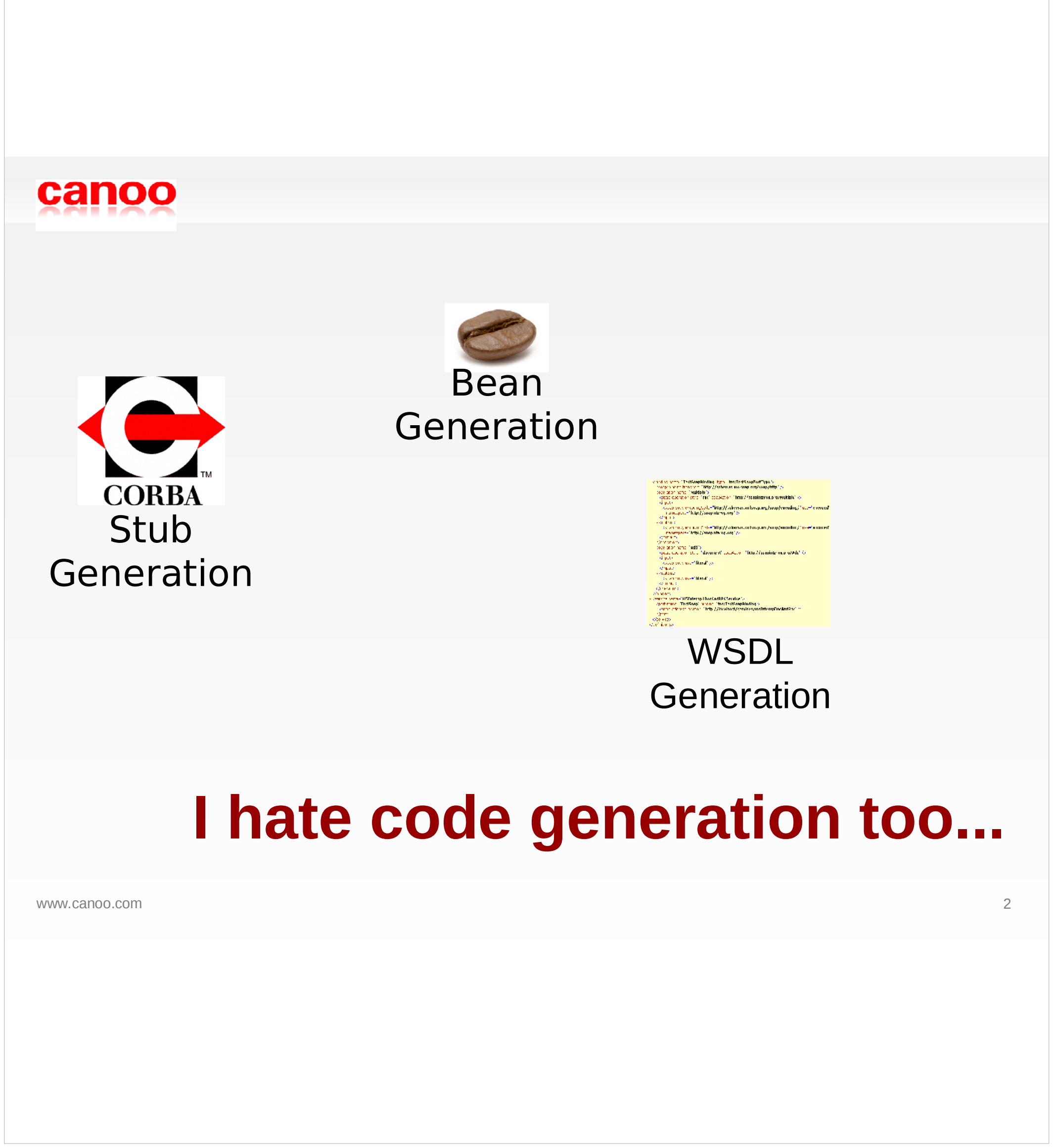
canoo › your provider for business web solutions ›

I work for Canoo. They paid for my trip to Cologne; in return I put their logo on all my slides. I would wear a Canoo shirt, but I am too fat and the shirt they gave me does not fit. Maybe by the end of the summer it will fit. Canoo is a very cool company, so I am sure they would give me a new shirt if I asked, but I would rather have the incentive to lose a little weight.

I keep a blog. If you like it leave a comment or give me an upvote, I find it rewarding.

I am a committer on Groovy and a few other projects. I just moved to Basel, so I have not had as much time for this as I'd like.

I am a JetBrains Academy member and can give away a few free licenses. W00t!



canoo

Bean
Generation

CORBA
Stub
Generation

WSDL
Generation

I hate code generation too...

www.canoo.com 2

Is compiler metaprogramming a bad idea?

Code generation truly sucks. Almost all programmers have experienced some sort of code generation hell with the above technologies.

Code generation looks good during the design process. It is great until you need to step through generated code in a debugger or research why it doesn't work or perform a full build in order to test your every little change.

Why do we have so much crummy generated code? I think the people selling generated code are frequently **not** the same people as those maintaining the generated code. In my opinion, if your designers are not your implementors then they invariably end up making your life hell.

… but it isn't all bad.

Is compiler metaprogramming a good idea?

Code generation is a strength of the Lisp and the Boo languages. Hopefully, I can convince you it is a strength of Groovy too.

What is compiler metaprogramming?

I define it as any sort of code generation that occurs before runtime: as part of a compiler, build process, or design phase. This is a broad definition, meant to make it seem not so extraordinary.

Good vs. Bad Compiler Metaprogramming?

All of the bad examples (previous) bring more code into the world. All of the good examples (here) result in less code overall. Good metaprogramming means there is less code. Period.

Project Lombok

```
import lombok.Getter;
import lombok.Setter;

public class Person {

  @Getter @Setter private String
firstName;
  @Setter @Setter private String
lastname;

}
```

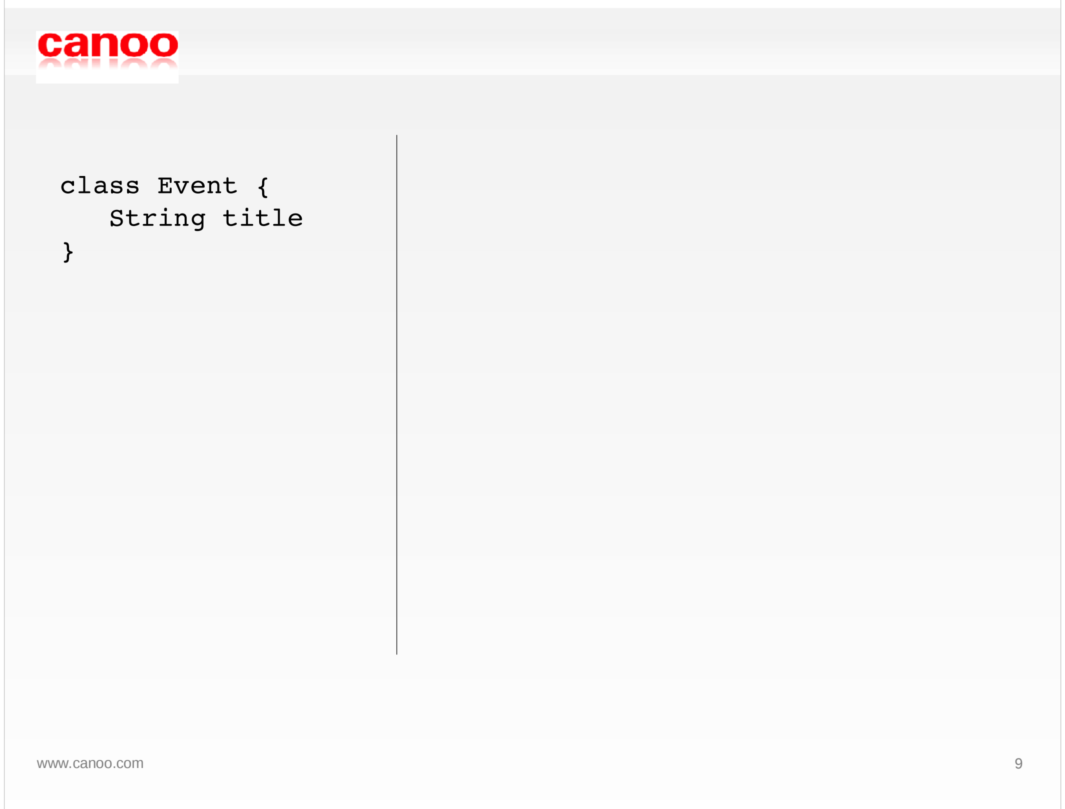
```
public class Person {
  private String firstName;
  private String lastName;

  void setFirstName(String fName) {
    this.firstName = fName;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setLastName(String lName
    this.lastName = lName;
  }

  public String getLastName() {
    return firstName;
  }
}
```

```
import lombok.Synchronized;

public class SynchronizedExample
{

    @Synchronized
    public void doSomething() {
        return ...;
    }
}
```

```
public class SynchronizedExample {
   private final Object $lock = new
Object[0];

   public void doSomething() {
      synchronized($lock) {
         return ...;
      }
   }
}
```

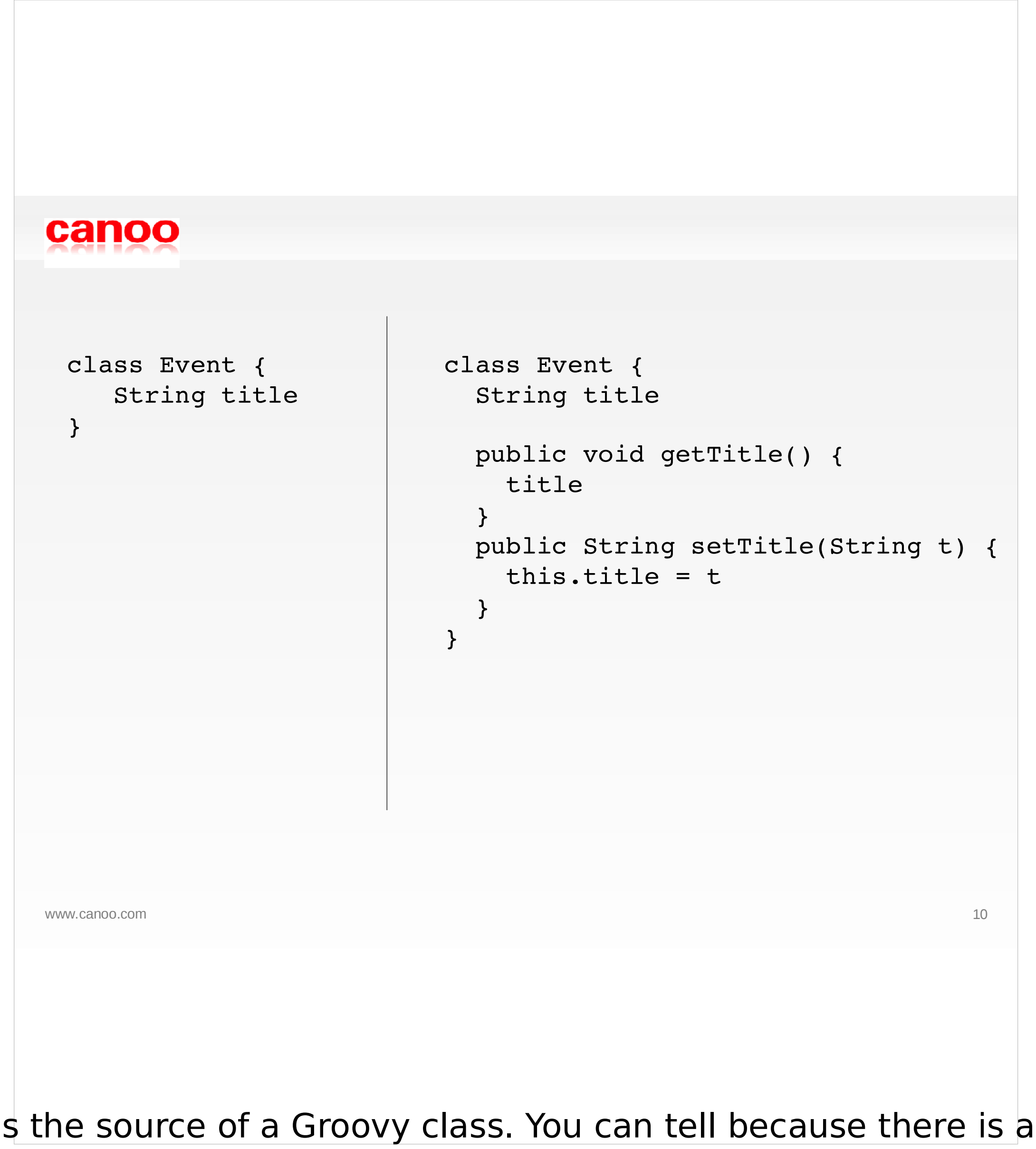
- Generates Java Boilerplate
- Compile Time Only
  - For Eclipse and javac
- Removable with delombok
- Read the fine print

  – You should know what is generated



```
class Event {
   String title
}
```

This is the source of a Groovy class. You can tell because there is a semi-colon missing. Mind-blowing, right?

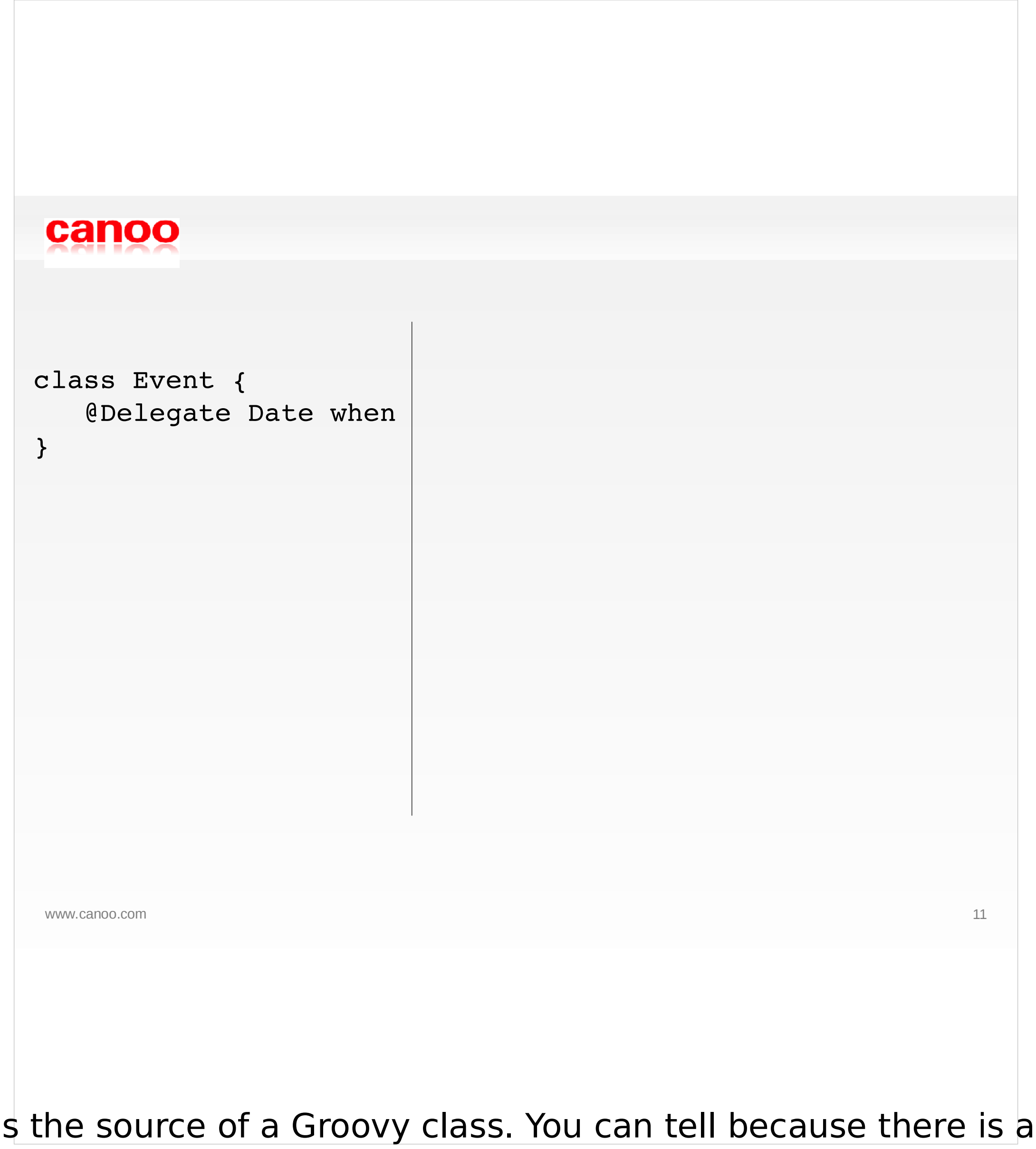


canoo

```
class Event {
   String title
}
```

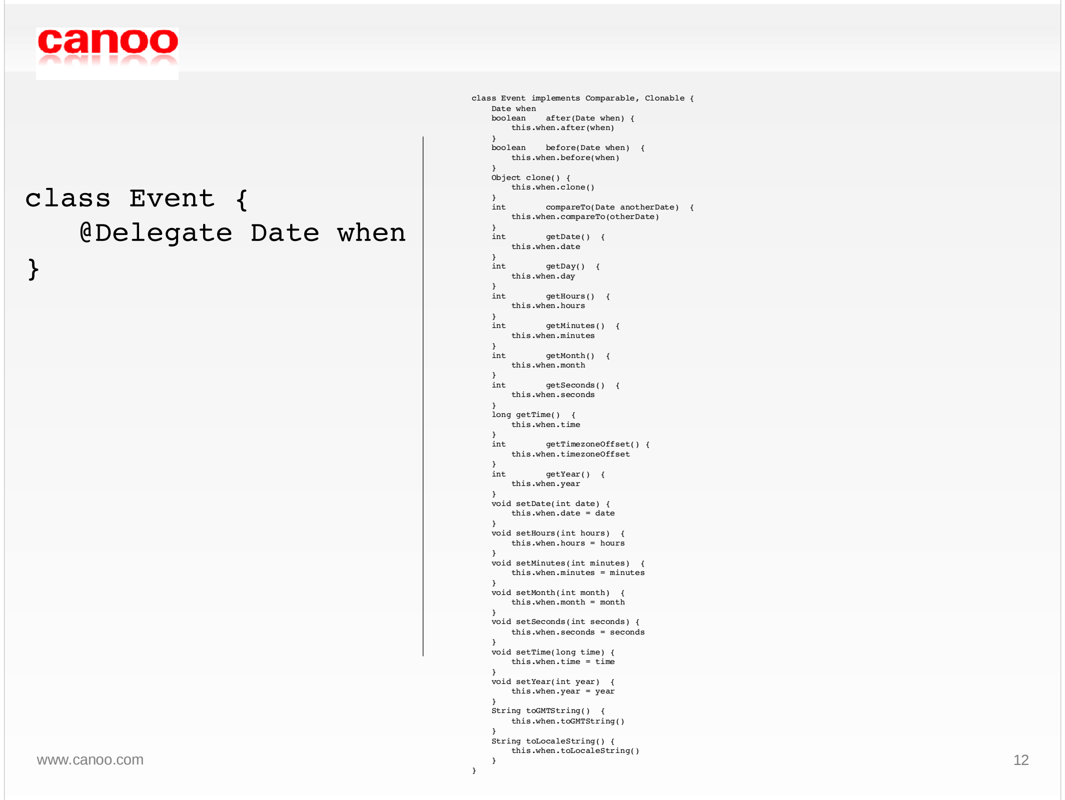
```
class Event {
  String title

  public void getTitle() {
    title
  }
  public String setTitle(String t) {
    this.title = t
  }
}
```

This is the source of a Groovy class. You can tell because there is a semi-colon missing. Mind-blowing, right?



```
class Event {
   @Delegate Date when
}
```

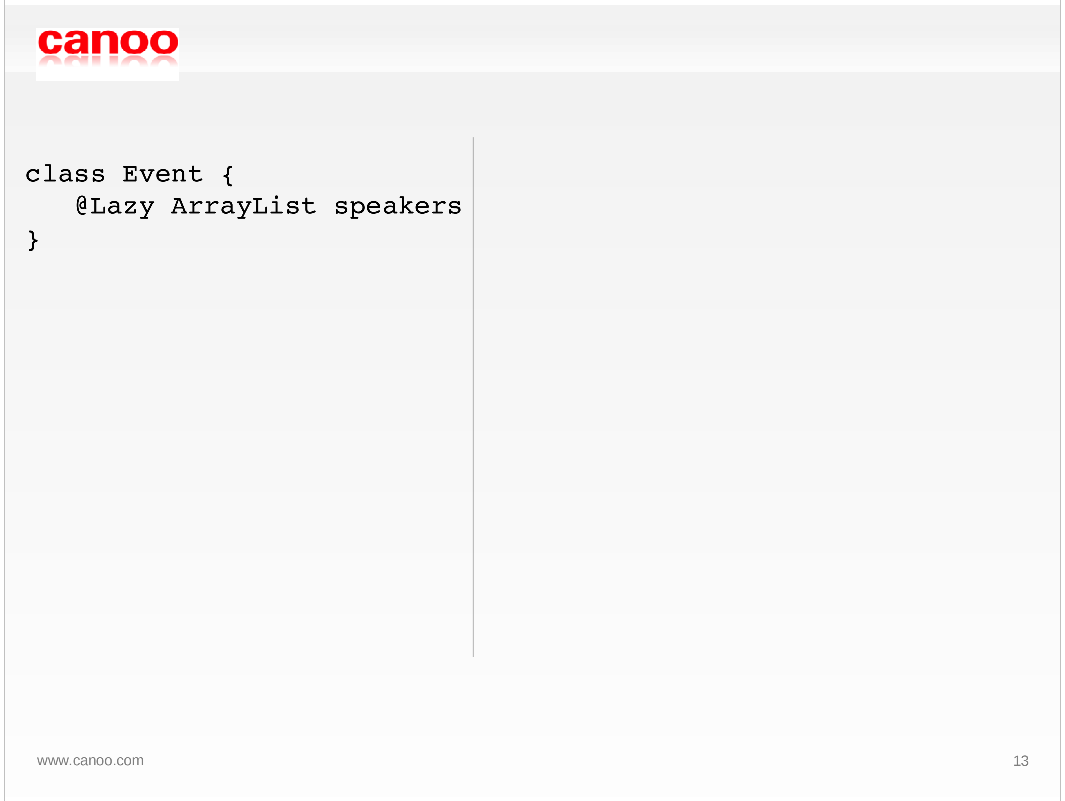
This is the source of a Groovy class. You can tell because there is a semi-colon missing. Mind-blowing, right?



```
class Event {
    @Delegate Date when
}
```

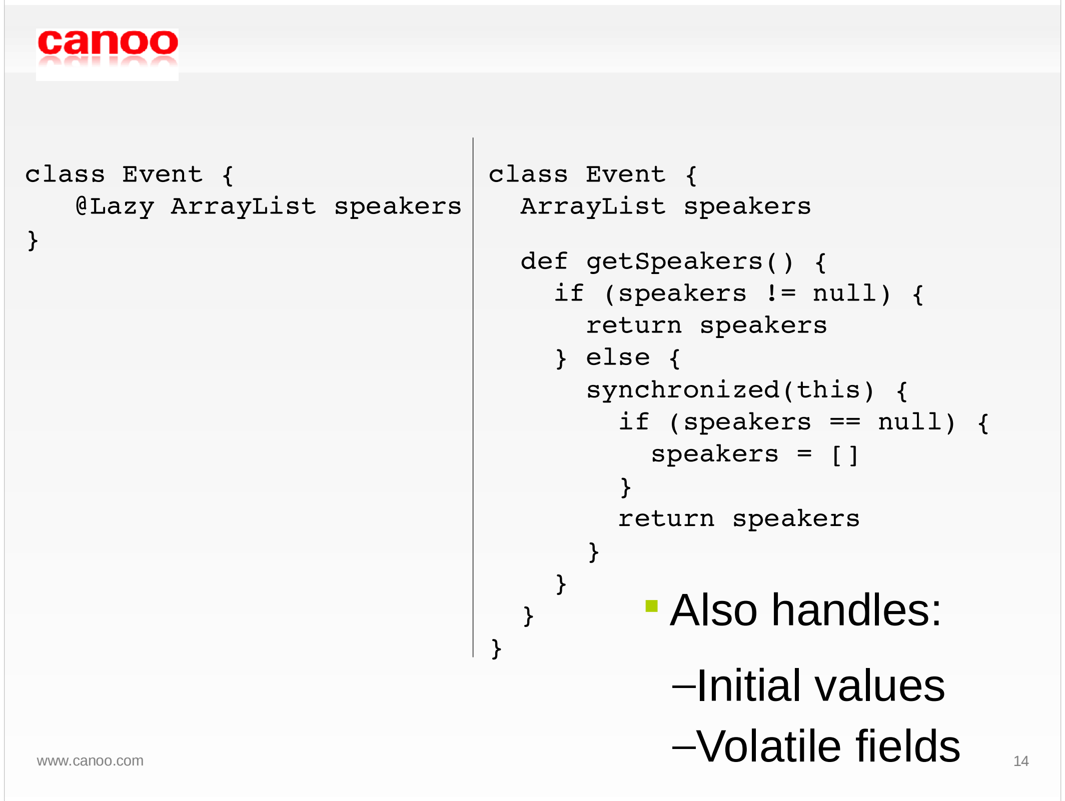
```
class Event implements Comparable, Clonable {
    Date when
    boolean    after(Date when) {
        this.when.after(when)
    }
    boolean    before(Date when)  {
        this.when.before(when)
    }
    Object clone() {
        this.when.clone()
    }
    int        compareTo(Date anotherDate)  {
        this.when.compareTo(otherDate)
    }
    int        getDate()  {
        this.when.date
    }
    int        getDay()  {
        this.when.day
    }
    int        getHours()  {
        this.when.hours
    }
    int        getMinutes()  {
        this.when.minutes
    }
    int        getMonth()  {
        this.when.month
    }
    int        getSeconds()  {
        this.when.seconds
    }
    long getTime()  {
        this.when.time
    }
    int        getTimezoneOffset() {
        this.when.timezoneOffset
    }
    int        getYear()  {
        this.when.year
    }
    void setDate(int date) {
        this.when.date = date
    }
    void setHours(int hours)  {
        this.when.hours = hours
    }
    void setMinutes(int minutes)  {
        this.when.minutes = minutes
    }
    void setMonth(int month)  {
        this.when.month = month
    }
    void setSeconds(int seconds) {
        this.when.seconds = seconds
    }
    void setTime(long time) {
        this.when.time = time
    }
    void setYear(int year)  {
        this.when.year = year
    }
    String toGMTString()  {
        this.when.toGMTString()
    }
    String toLocaleString() {
        this.when.toLocaleString()
    }
}
```

This is the source of a Groovy class. You can tell because there is a semi-colon missing. Mind-blowing, right?



```
class Event {
   @Lazy ArrayList speakers
}
```

This is the source of a Groovy class. You can tell because there is a semi-colon missing. Mind-blowing, right?



```
class Event {
   @Lazy ArrayList speakers
}
```

```
class Event {
  ArrayList speakers

  def getSpeakers() {
    if (speakers != null) {
      return speakers
    } else {
      synchronized(this) {
        if (speakers == null) {
          speakers = []
        }
        return speakers
      }
    }
  }
}
```

- Also handles:
  - –Initial values
  - –Volatile fields

This is the source of a Groovy class. You can tell because there is a semi-colon missing. Mind-blowing, right?

```
@Immutable
class Event {
   String title
}
```

```
@Immutable
class Event {
   String title
}
```

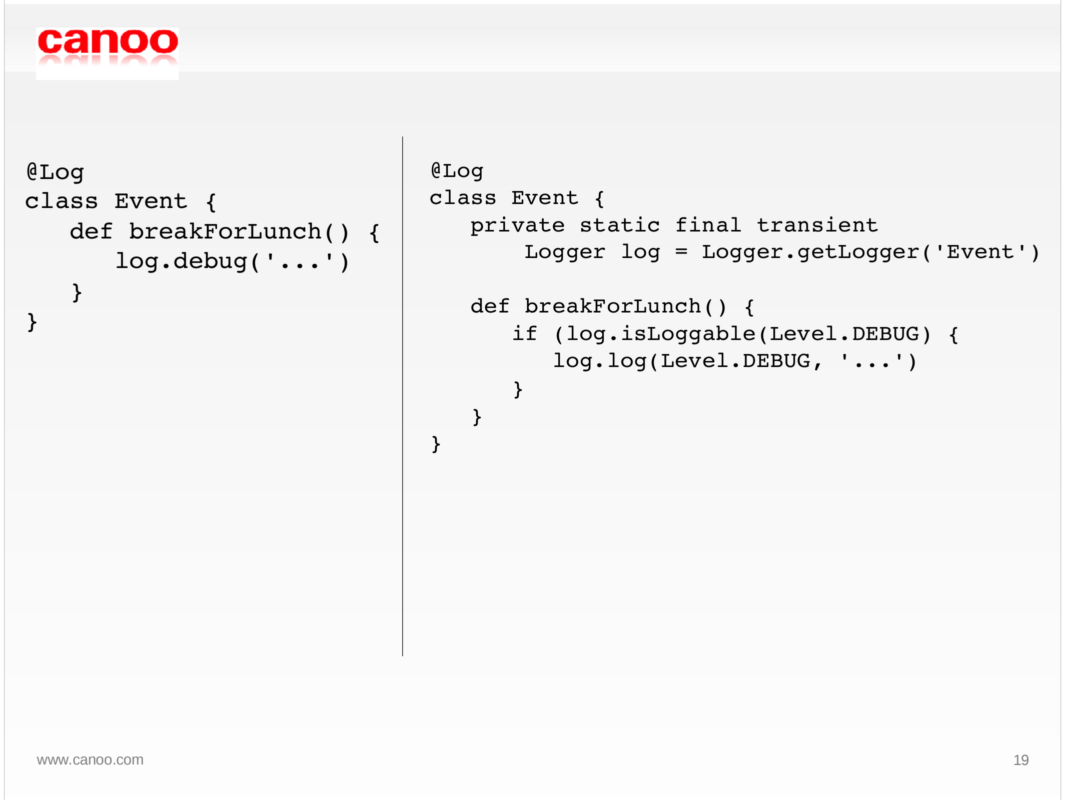
- Class is final
- Properties must be @Immutable or effectively immutable
- Properties are private
- Mutatators throw ReadOnlyPropertyException
- Map constructor created
- Tuple constructor created
- Equals(), hashCode() and toString() created
- Dates, Clonables, and arrays are defensively copied on way in and out (but not deeply cloned)
- Collections and Maps are wrapped in Immutable variants
- Non-immutable fields force an error
- Special handling for Date, Color, etc
- Many generated methods configurable

@Newify
@Category
@Package Scope
@Grab

… and many more as libraries

```
@Log
class Event {
   def breakForLunch() {
      log.debug('...')
   }
}
```

This is the source of a Groovy class. You can tell because there is a semi-colon missing. Mind-blowing, right?



```
@Log
class Event {
   def breakForLunch() {
      log.debug('...')
   }
}
```

```
@Log
class Event {
   private static final transient
       Logger log = Logger.getLogger('Event')

   def breakForLunch() {
      if (log.isLoggable(Level.DEBUG) {
         log.log(Level.DEBUG, '...')
      }
   }
}
```

This is the source of a Groovy class. You can tell because there is a semi-colon missing. Mind-blowing, right?

canoo

Coming in Groovy 1.8

@Log

@Synchronized

@IndexedProperty

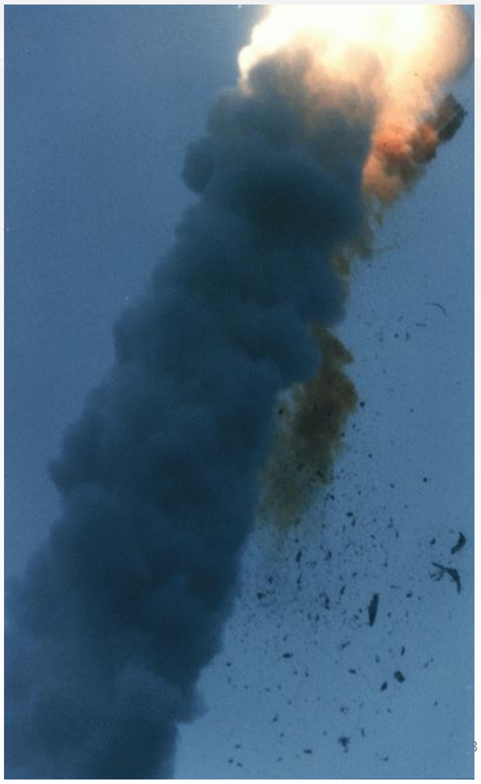
@InheritConstructors

cité
de
l'espace
arianespace
esa
esa
cnes
cnes

cité
de
l'espace
arianespace
esa
esa
cnes
cnes

Ariane 5 Destroyed
US$370 *million* in damages

… from a integer overflow error



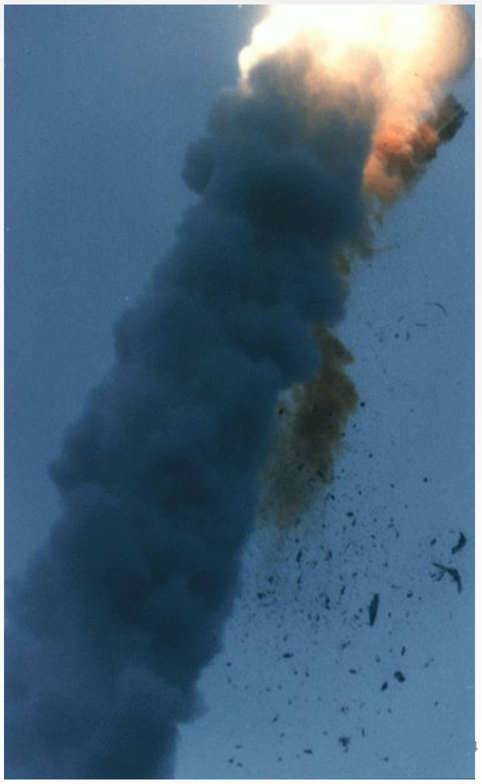
canoo

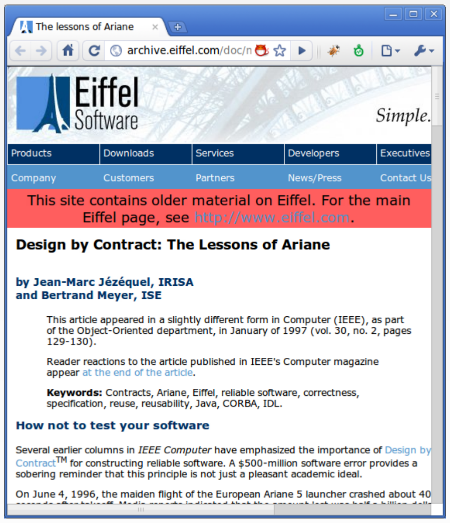
Ariane 5 Destroyed
US$370 *million* in damages

… from a integer overflow error

Greece bailout is $145 *billion*

… so we look quite good
compared to the bankers.





This site contains older material on Eiffel. For the main Eiffel page, see http://www.eiffel.com.

Design by Contract: The Lessons of Ariane

by Jean-Marc Jézéquel, IRISA
and Bertrand Meyer, ISE

This article appeared in a slightly different form in Computer (IEEE), as part of the Object-Oriented department, in January of 1997 (vol. 30, no. 2, pages 129-130).

Reader reactions to the article published in IEEE's Computer magazine appear at the end of the article.

Keywords: Contracts, Ariane, Eiffel, reliable software, correctness, specification, reuse, reusability, Java, CORBA, IDL.

How not to test your software

Several earlier columns in *IEEE Computer* have emphasized the importance of Design by Contract™ for constructing reliable software. A $500-million software error provides a sobering reminder that this principle is not just a pleasant academic ideal.

On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40

“Design by Contract and Eiffel would have automatically avoided the crash...”

Sincerely,

The Eiffel Guys

(not a direct quote)

DEMO

hooray

Design by Contract™

Component semantics part of interface

Semantics enforced by implementation

Contracts describe valid test results

Self Documenting

Sound good?

canoo

Eiffel Envy

Symptoms include:

I exhibit 4 out of 5 of these. You are free to guess which one does not apply to me.

Eiffel Envy

Symptoms include:

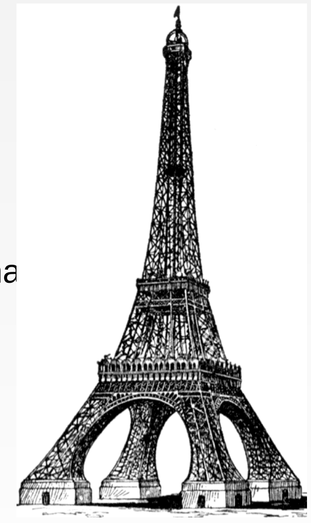
Unhealthy fixation on correctness

Pedantic use of unit tests

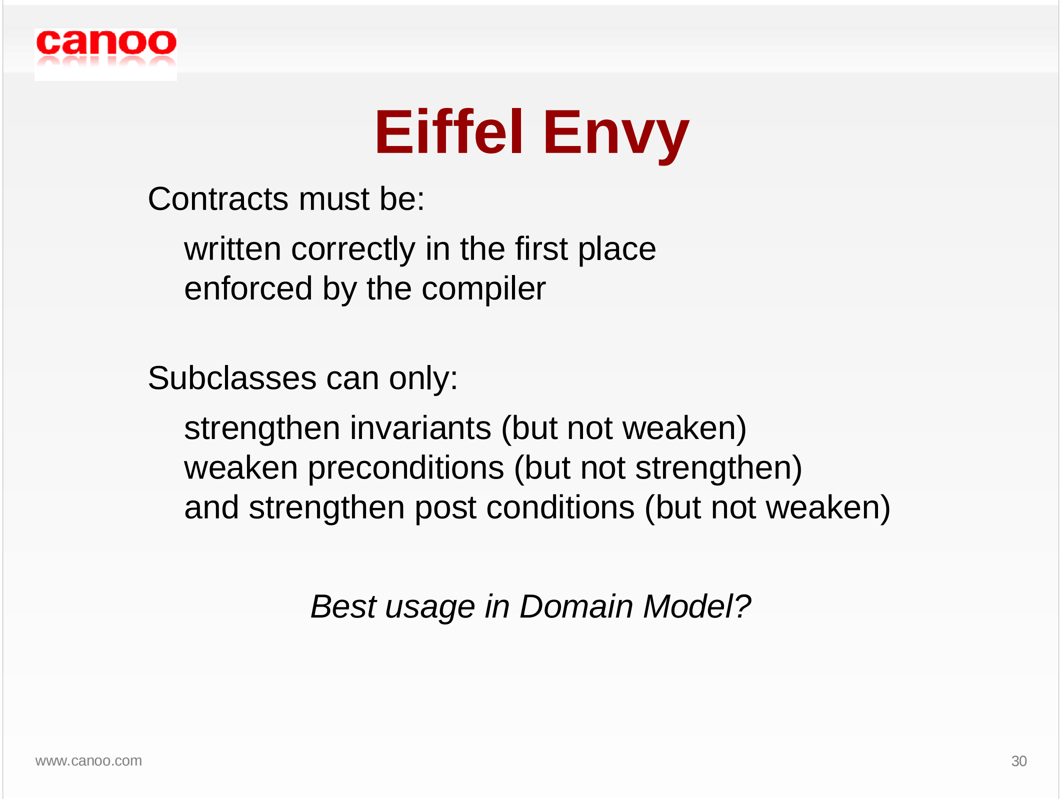
Domineering mothers

Having a 3 page interview questionnaire testing Java arcana

Dreams of being chased by goats



I exhibit 4 out of 5 of these. You are free to guess which one does not apply to me.



Eiffel Envy

Contracts must be:

- written correctly in the first place
- enforced by the compiler

Subclasses can only:

- strengthen invariants (but not weaken)
- weaken preconditions (but not strengthen)
- and strengthen post conditions (but not weaken)

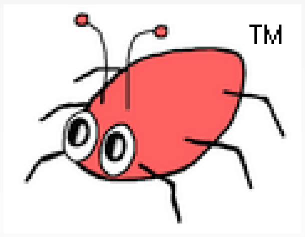
Best usage in Domain Model?

I exhibit 4 out of 5 of these. You are free to guess which one does not apply to me.

Q. Which OS Project won a 2008 Jolt Award for lamest logo?

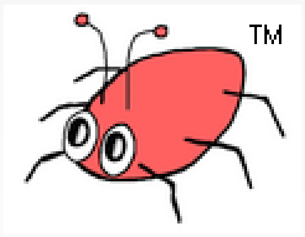
Q. Which OS Project won a 2008 Jolt Award for lamest logo?

A. FindBugs



Q. Which OS Project won a 2008 Jolt Award for lamest logo?

A. FindBugs



P.S. His name is “Buggy” and he is trademarked so no one steals him

FindBugs Finds Bugs

Empty synchronized block
Inconsistent synchronization
Synchronization on Boolean could lead to deadlock
Synchronization on boxed primitive could lead to deadlock
Synchronization on interned String could lead to deadlock

… and 364 more rules

FindBugs Finds Bugs

Empty synchronized block
Inconsistent synchronization
Synchronization on Boolean could lead to deadlock
Synchronization on boxed primitive could lead to deadlock
Synchronization on interned String could lead to deadlock

… and 364 more rules

CodeNarc Finds Bugs

ThreadLocal not static final field
Volatile long or double field
Nested synchronization
Synchronized method, synchronized on this
Call to System.runFinalizersOnExit()

… and 67 more rules

FindBugs Finds Bugs

Empty synchronized block
Inconsistent synchronization
Synchronization on Boolean could lead to deadlock
Synchronization on boxed primitive could lead to deadlock
Synchronization on interned String could lead to deadlock

… and 364 more rules

CodeNarc Finds Bugs

ThreadLocal not static final field
Volatile long or double field
Nested synchronization
Synchronized method, synchronized on this
Call to System.runFinalizersOnExit()

… and 67 more rules

Embedded Languages

```
def s = new ArithmeticShell()

assert 2 == s.evaluate(' 1+1 ')
assert 1.0 == s.evaluate('cos(2*PI)')
```

… source in groovy/src/examples/groovyShell

Embedded Languages

```
def s = new ArithmeticShell()

assert 2 == s.evaluate(' 1+1 ')
assert 1.0 == s.evaluate('cos(2*PI)')

shouldFail(SecurityException) {
    s.evaluate('new File()')
}
```

… source in groovy/src/examples/groovyShell

Embedded Languages

Tired of hearing about DSLs?

Say “Embedded Language” instead!

ArithmeticShell ~= 300 lines of code

Alternative is

- javacc?
- custom interpreter?

Embedded Languages

Tired of hearing about DSLs?

Say “Embedded Language” instead!

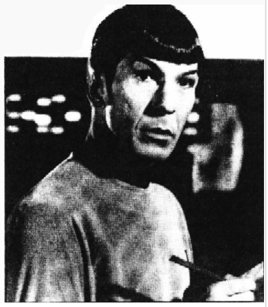
ArithmeticShell ~= 300 lines of code

Alternative is

- javacc?
- custom interpreter?
- seppuku?

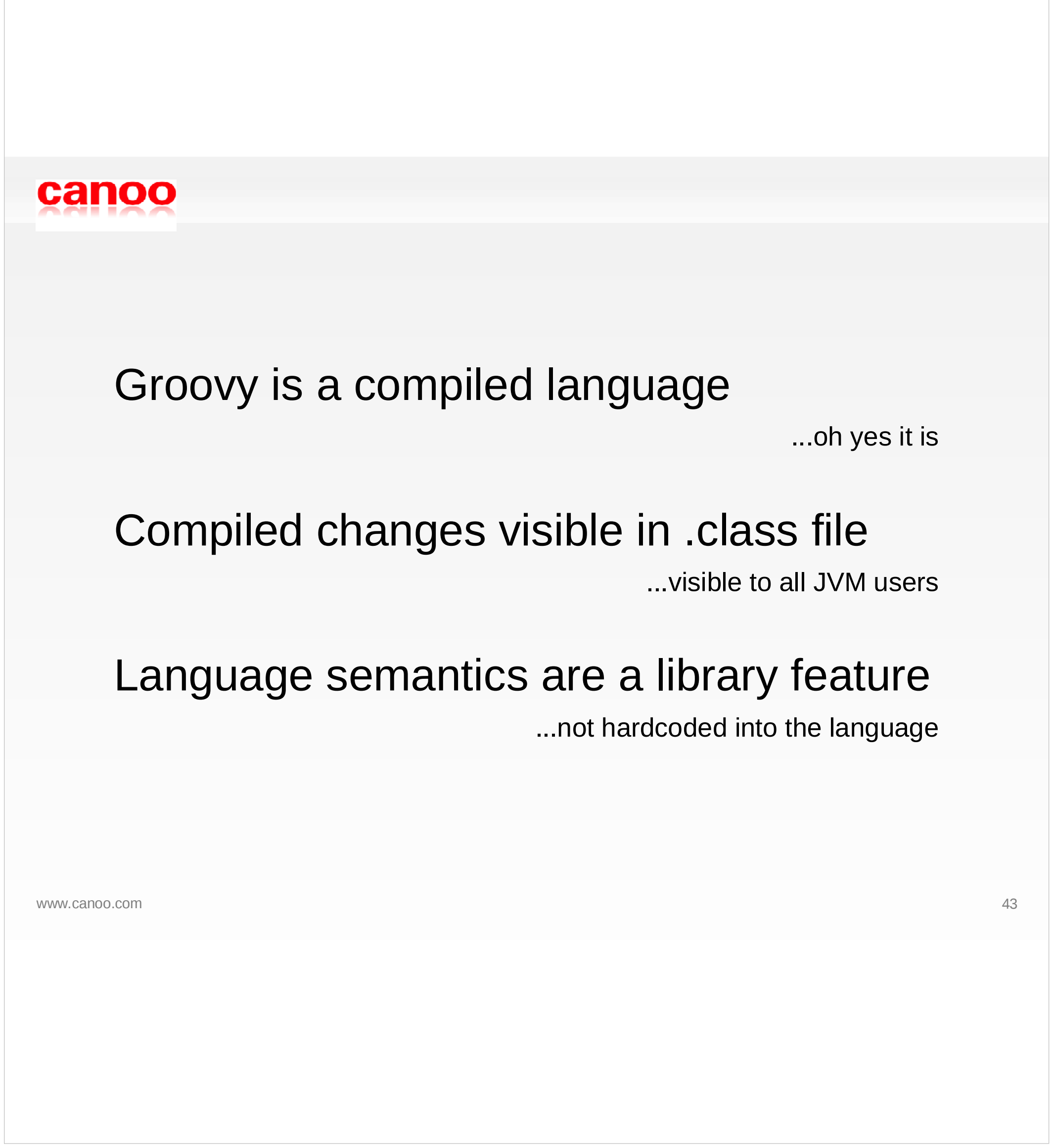
Java Perversions

```
def "Does simple math work?"() {
    expect:
    def s = new ArithmeticShell()
    s.evaluate(input) == output

    where:
    input              | output
    '1 + 1'            | 2
    'cos(2*PI)'        | 1.0
}
```



DEMO

again?



Groovy is a compiled language

...oh yes it is

Compiled changes visible in .class file

...visible to all JVM users

Language semantics are a library feature

...not hardcoded into the language

1. Even if you don't invoke groovyc, your Groovy scripts are being compiled. Even if .class files are not written to your hard disk, your Groovy class is being transformed into a Java Class file.

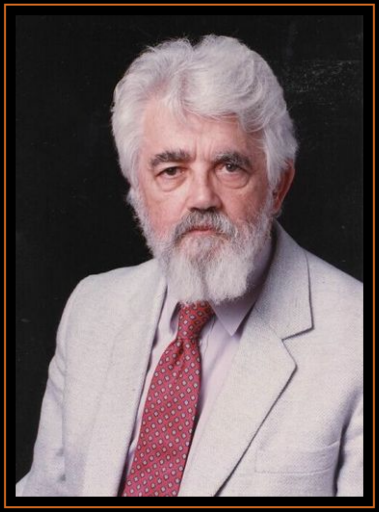
2. If you make changes that are written into the .class file by Groovy, then those changes are visible when the code is called from Java, Scala, Clojure, or “bf for Java”. This is not the case with runtime metaprogramming like metaclasses and invokeMethod/invokeProperty

3. The semantics of the Groovy language is now a feature that can be modified by you: the framework and library writers. To a more limited extent, the Groovy syntax of Groovy is extensible as well.

canoo

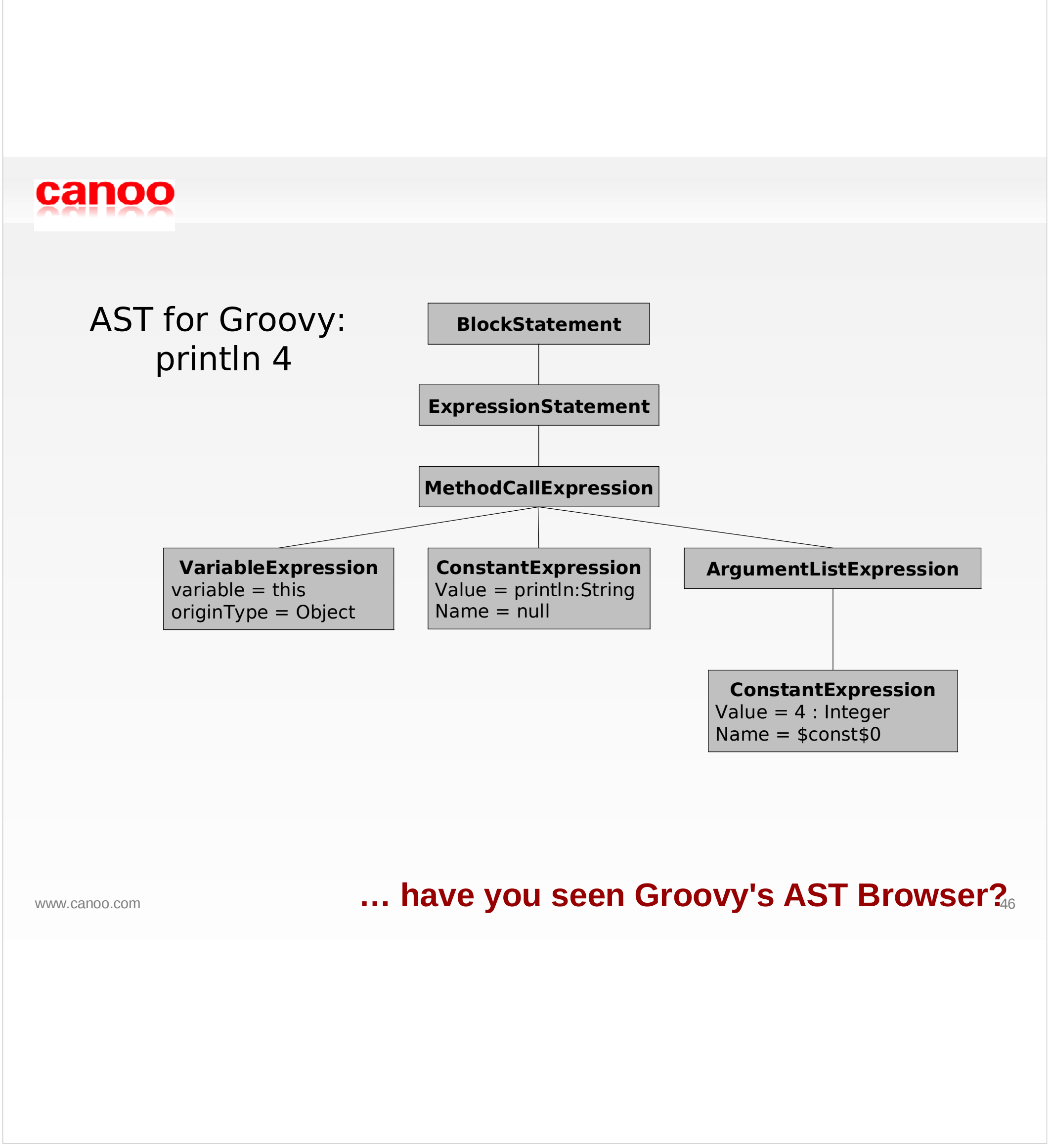
"A language should have access to its own abstract syntax"

John McCarthy

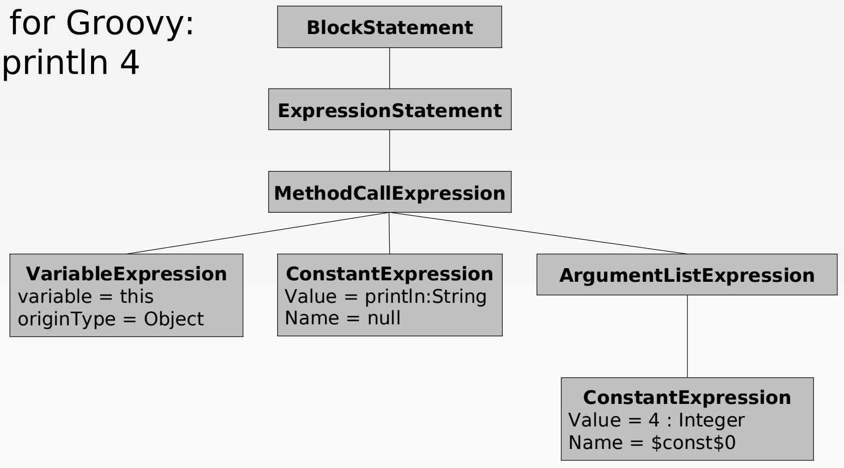


PROGRAMMING

You're doing it completely wrong.



AST for Groovy: println 4



www.canoo.com

… have you seen Groovy's AST Browser?

46

“AST Transformations” is the name of the Groovy feature that lets you do these things.

AST is an abstract syntax tree: a tree with leaves and branches representing the source code. Most languages transform source into an AST. The Eclipse Java compiler transforms Java into AST, and Project Lombock is based off of this. IntelliJ IDEA transforms Java into AST, and many IDEA plugins are based off of this.

Groovy lets you view and modify the AST. You can add things into the tree, you can remove them, you can observe them.

GroovyConsole has an AST Browser that lets you view AST. This is a key tool to understanding AST. If you are going to write an AST Transformation then start by using GroovyConsole's AST Browser to understand what you need to do.

DEMO

OH YEAH

How It Works

Local AST Transformations

Global AST Transformations

AST Builder

AST Templates

ANTLR Plugins

@Requires(...)
...
source.groovy

```
@Requires(...)
...
```

source.groovy

```
public @interface Requires {
    ...
}
```

Requires.java

```
@Requires(...)
...
```

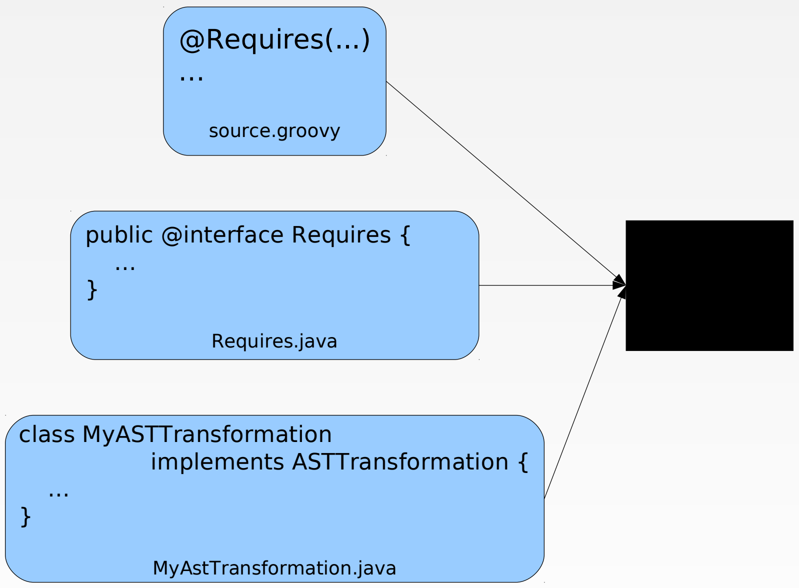
source.groovy

```
public @interface Requires {
    ...
}
```

Requires.java

```
class MyASTTransformation
                  implements ASTTransformation {
    ...
}
```

MyAstTransformation.java



```
@Requires(...)
...
```

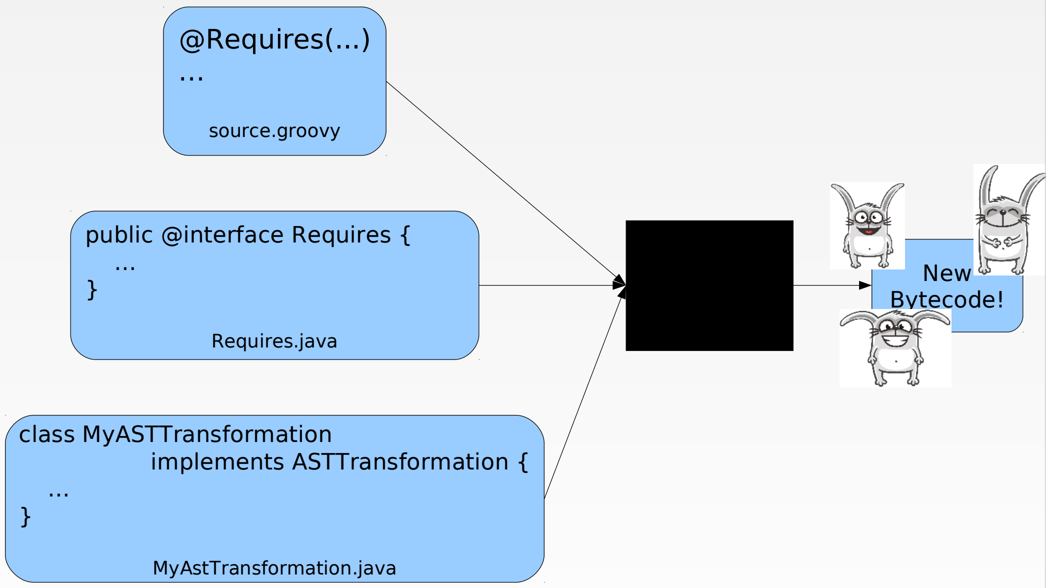
source.groovy

```
public @interface Requires {
    ...
}
```

Requires.java

```
class MyASTTransformation
                implements ASTTransformation {
    ...
}
```

MyAstTransformation.java



```
@Requires(...)
...
```

```
public @interface Requires {
    ...
}
```

```
class MyASTTransformation
              implements ASTTransformation {
   ...
}
```

```
@Requires(...)
void startEngine() { ... }
```

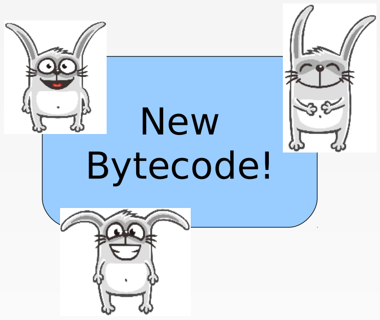
```
@Requires(...)
void startEngine() { ... }
```

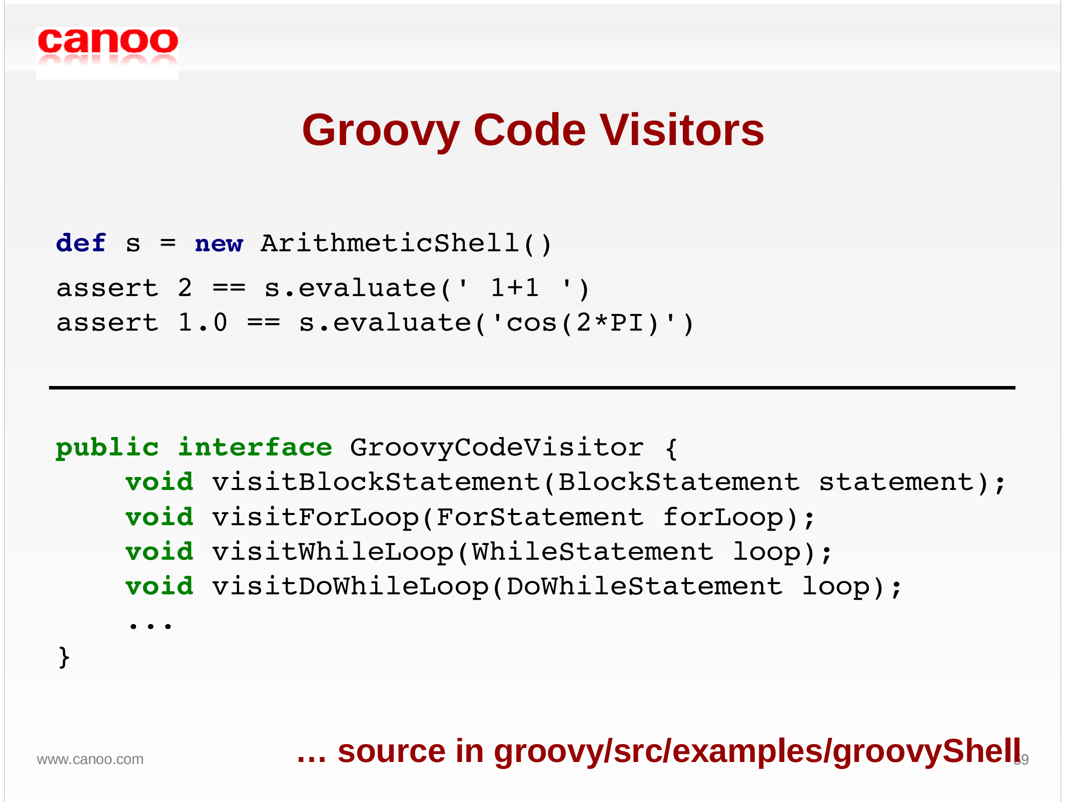
```
@GroovyASTTransformationClass("org.pkg.MyTransformation")
public @interface Requires {
    ...
}
```

```
@Requires(...)
void startEngine() { ... }
```

```
@GroovyASTTransformationClass("org.pkg.MyTransformation")
public @interface Requires {
    ...
}
```

```
@GroovyASTTransformation(phase = CompilePhase.CANONICALIZATION)
public class MyTransformation implements ASTTransformation {
    public void visit(ASTNode[] nodes, SourceUnit source) {
        ...
    }
}
```


New Bytecode!



Groovy Code Visitors

```
def s = new ArithmeticShell()
assert 2 == s.evaluate(' 1+1 ')
assert 1.0 == s.evaluate('cos(2*PI)')
```

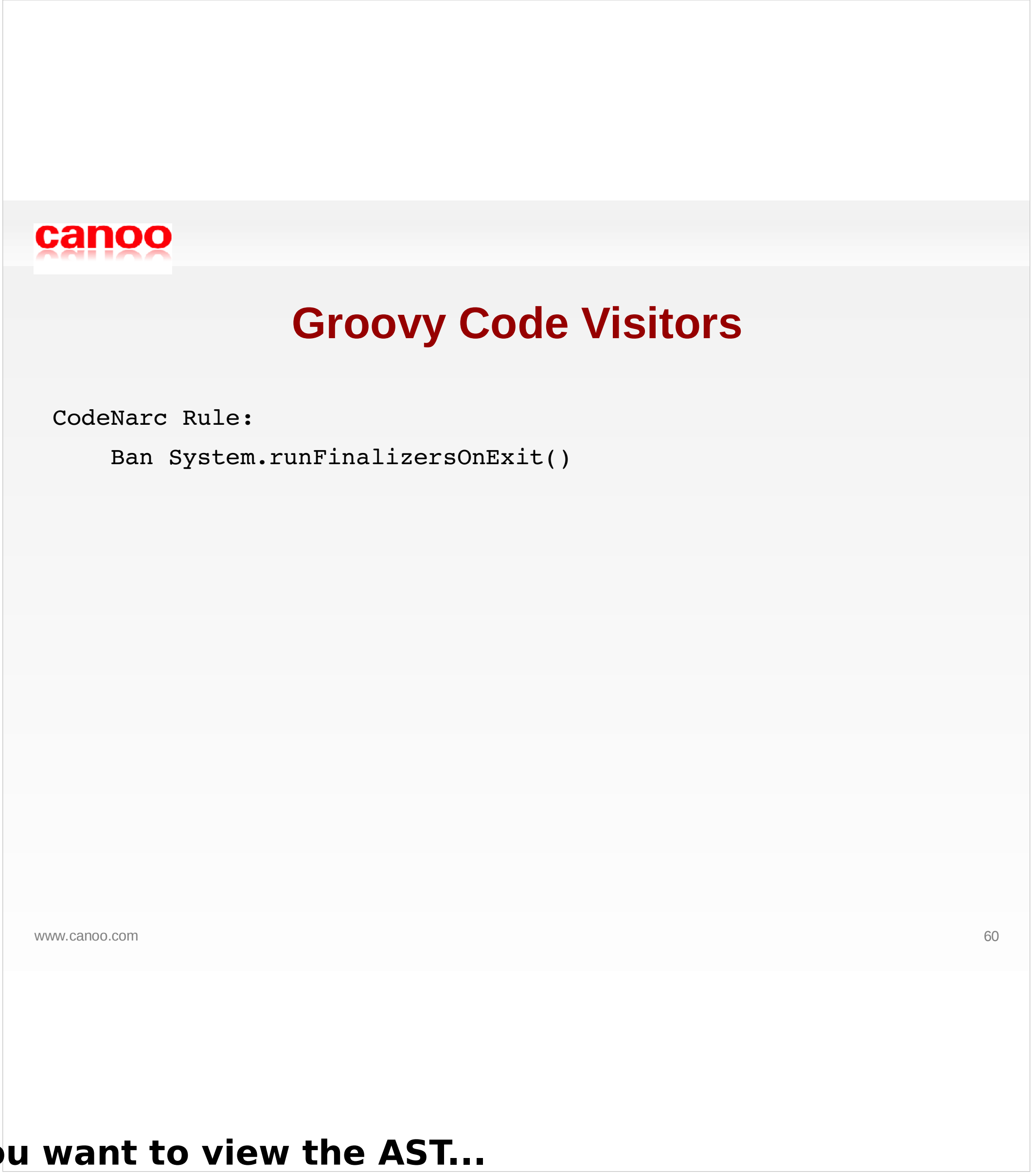
```
public interface GroovyCodeVisitor {
    void visitBlockStatement(BlockStatement statement);
    void visitForLoop(ForStatement forLoop);
    void visitWhileLoop(WhileStatement loop);
    void visitDoWhileLoop(DoWhileStatement loop);
    ...
}
```

... source in groovy/src/examples/groovyShell

So you want to view the AST...

Groovy gives you several Visitor interfaces and adapters that allow you to observe the AST of a SourceUnit. The simplest is GroovyCodeVisitor: an interface with one method for each language feature in Groovy.

A Visitor is used in the ArithmeticShell. There is a whitelist of allowed Groovy features, objects, and packages. Each script is compiled when it is passed to ArithmeticShell (or GroovyShell). A SecurityException is raised if operations *not* within the allowed whitelist are found during the compile phase.



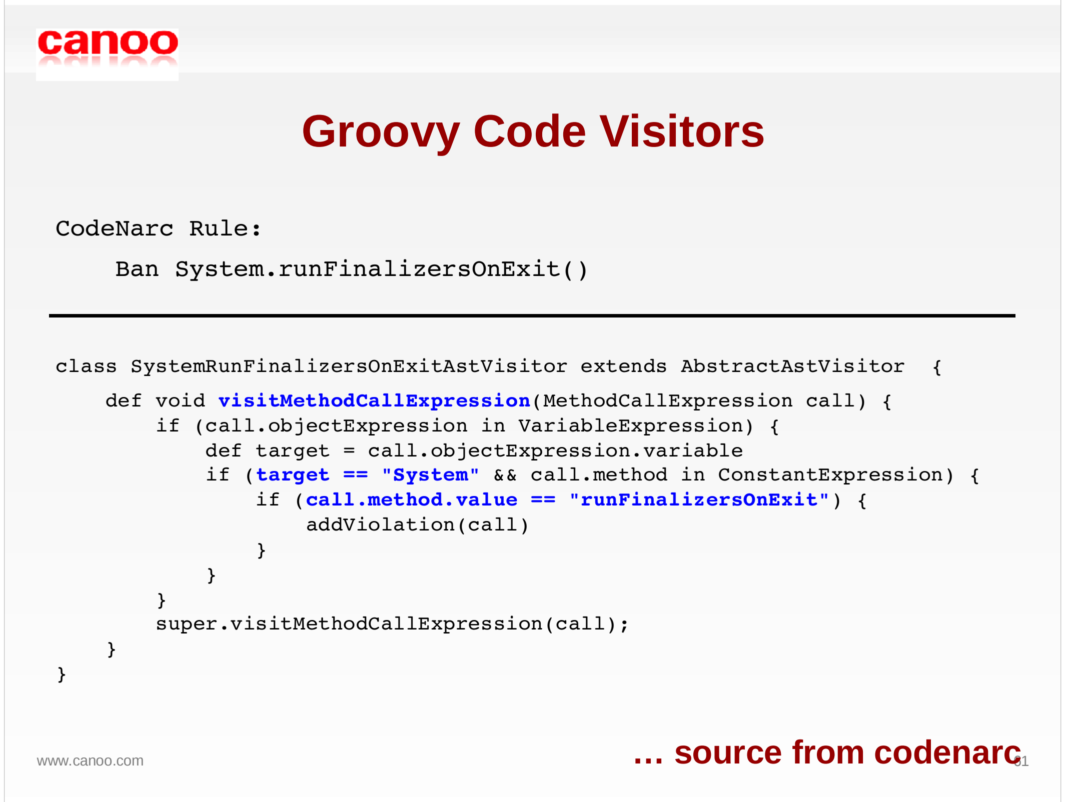
Groovy Code Visitors

```
CodeNarc Rule:
    Ban System.runFinalizersOnExit()
```

So you want to view the AST...

Groovy gives you several Visitor interfaces and adapters that allow you to observe the AST of a SourceUnit. The simplest is GroovyCodeVisitor: an interface with one method for each language feature in Groovy.

A Visitor is used in the ArithmeticShell. There is a whitelist of allowed Groovy features, objects, and packages. Each script is compiled when it is passed to ArithmeticShell (or GroovyShell). A SecurityException is raised if operations *not* within the allowed whitelist are found during the compile phase.



Groovy Code Visitors

```
CodeNarc Rule:
    Ban System.runFinalizersOnExit()
```

```
class SystemRunFinalizersOnExitAstVisitor extends AbstractAstVisitor  {
    def void visitMethodCallExpression(MethodCallExpression call) {
        if (call.objectExpression in VariableExpression) {
            def target = call.objectExpression.variable
            if (target == "System" && call.method in ConstantExpression) {
                if (call.method.value == "runFinalizersOnExit") {
                    addViolation(call)
                }
            }
        }
        super.visitMethodCallExpression(call);
    }
}
```

… source from codenarc

So you want to view the AST...

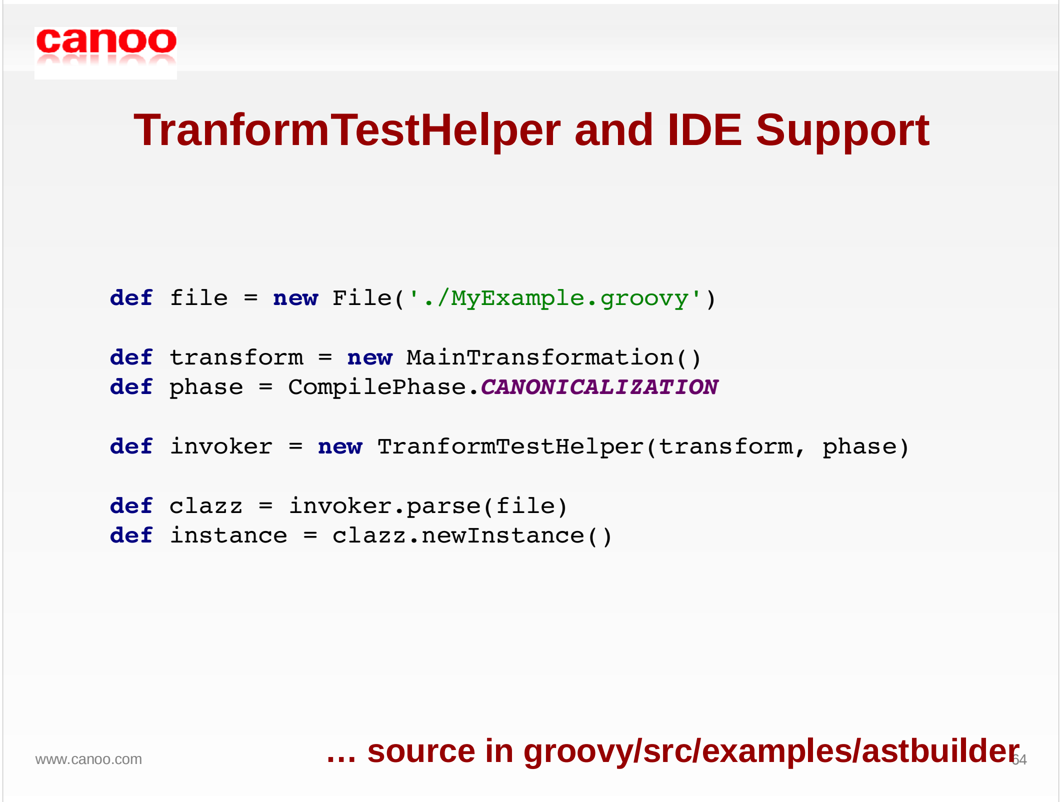
Groovy gives you several Visitor interfaces and adapters that allow you to observe the AST of a SourceUnit. The simplest is GroovyCodeVisitor: an interface with one method for each language feature in Groovy.

A Visitor is used in the ArithmeticShell. There is a whitelist of allowed Groovy features, objects, and packages. Each script is compiled when it is passed to ArithmeticShell (or GroovyShell). A SecurityException is raised if operations *not* within the allowed whitelist are found during the compile phase.

Pitfalls!

Pitfalls!

Testing AST Transformations
Writing AST
Rigid Groovy/Java syntax
Splicing source into AST
Finding insertion points
Splicing AST into source
Variable capture



TranformTestHelper and IDE Support

```
def file = new File('./MyExample.groovy')

def transform = new MainTransformation()
def phase = CompilePhase.CANONICALIZATION

def invoker = new TranformTestHelper(transform, phase)

def clazz = invoker.parse(file)
def instance = clazz.newInstance()
```

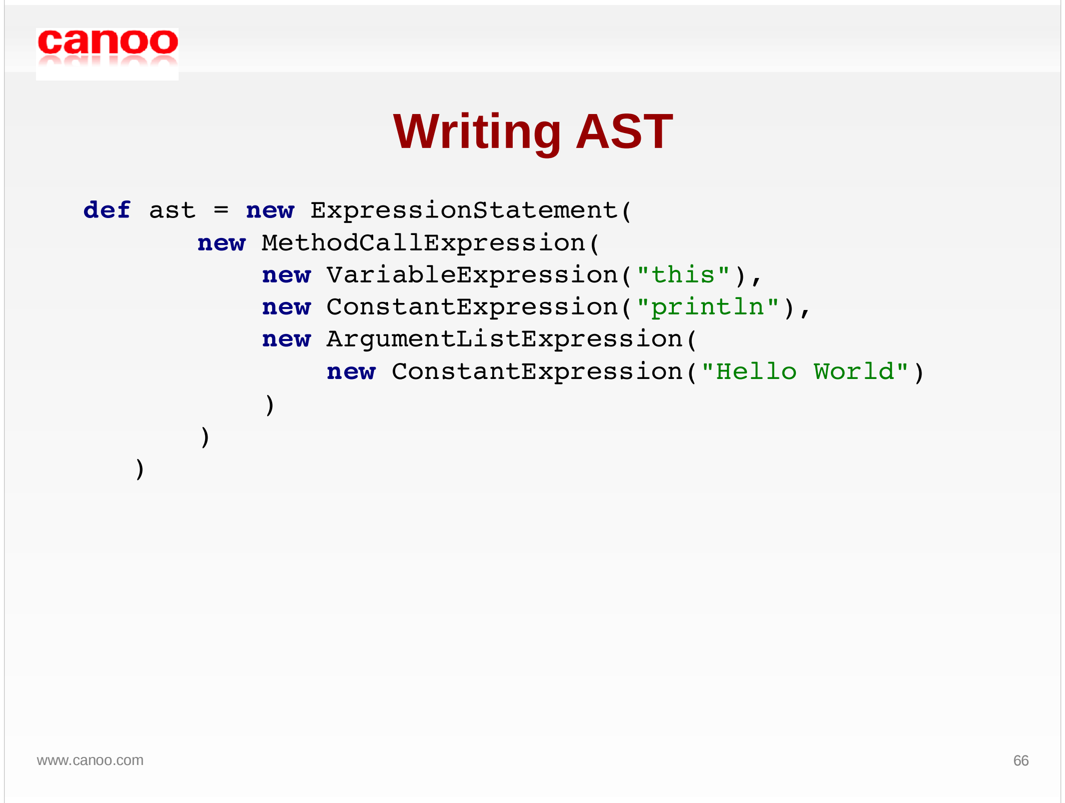
… source in groovy/src/examples/astbuilder

If you think you can write an AST Transformation without extensive unit tests then you are either wrong or Donald Knuth. And you ain't no Knuth.

Here are some must have tools for writing AST Transforms:
* AST Browser – to understand the AST you want to create
* IDE – to understand the AST you are trying to modify
* TransformTestHelper – to run your Transformation during test time
* Tons of Unit Tests – Really, truly try to break your transform with tests. Did you test inner classes? Anonymous classes? Nested classes? Import aliases? Static imports?

Pitfalls!

~~Testing AST Transformations~~
Writing AST
Rigid Groovy/Java syntax
Splicing source into AST
Finding insertion points
Splicing AST into source
Variable capture



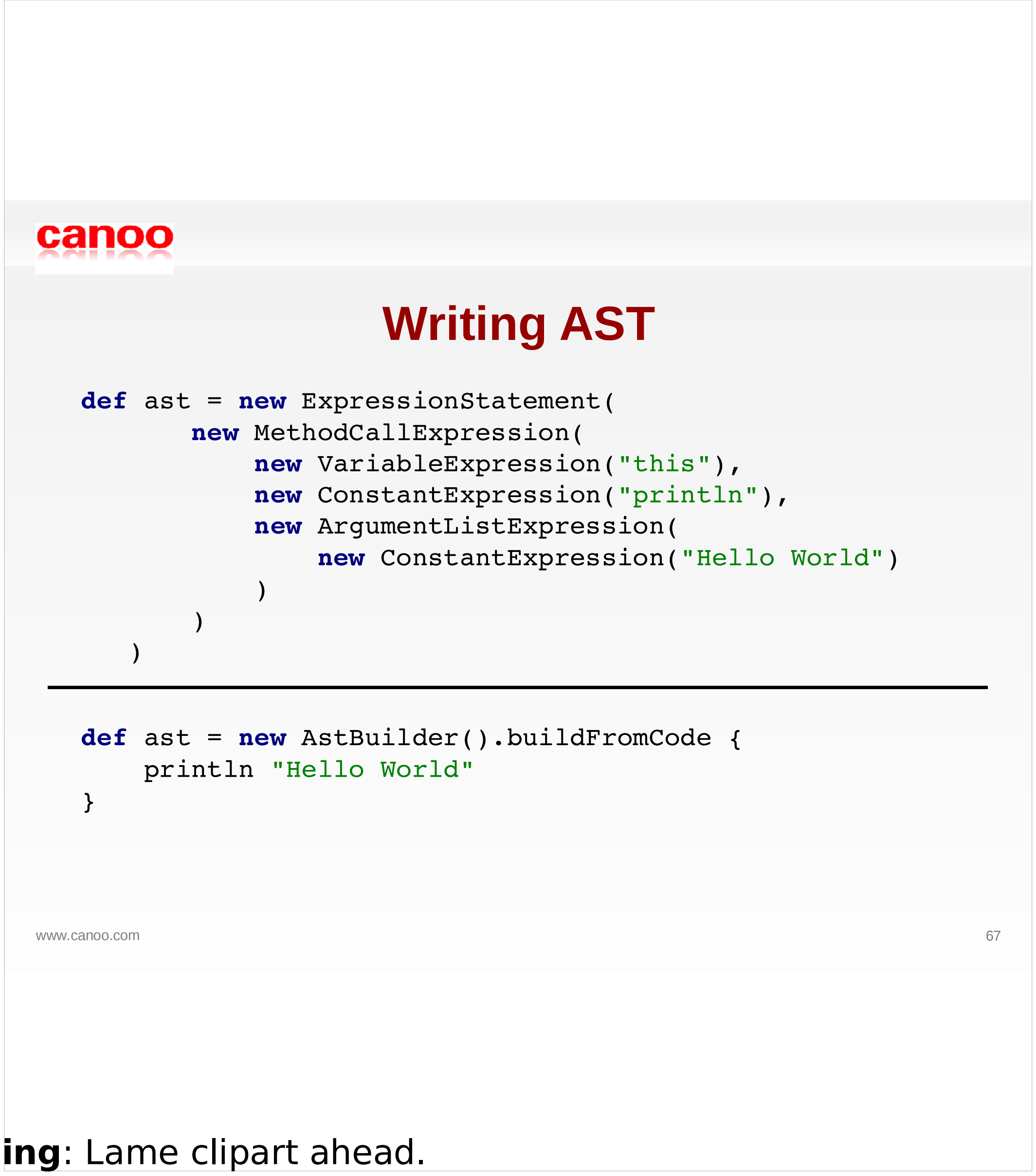
Writing AST

```
def ast = new ExpressionStatement(
       new MethodCallExpression(
           new VariableExpression("this"),
           new ConstantExpression("println"),
           new ArgumentListExpression(
               new ConstantExpression("Hello World")
           )
       )
   )
```

Warning: Lame clipart ahead.

The subtypes of ASTNode create a large and complex hierarchy. Writing AST by hand is verbose and unbearable (especially without an IDE).

You can do it but it is a pain.



Writing AST

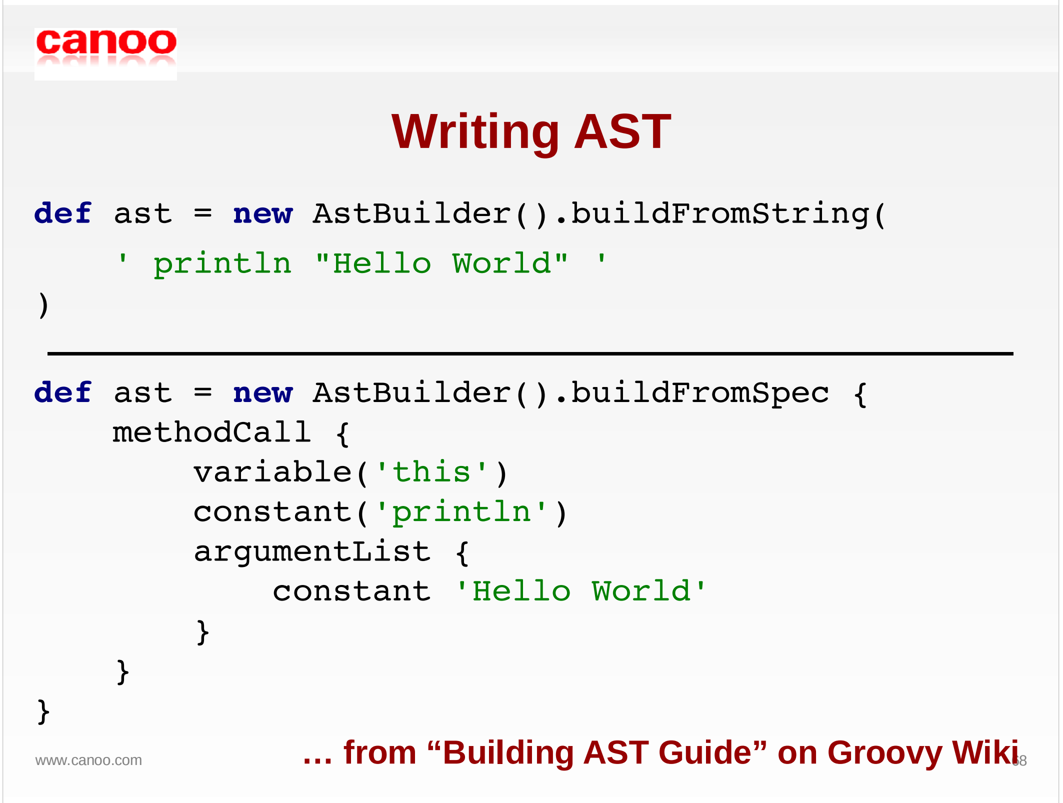
```
def ast = new ExpressionStatement(
        new MethodCallExpression(
            new VariableExpression("this"),
            new ConstantExpression("println"),
            new ArgumentListExpression(
                new ConstantExpression("Hello World")
            )
        )
    )
```

```
def ast = new AstBuilder().buildFromCode {
    println "Hello World"
}
```

Warning: Lame clipart ahead.

The subtypes of ASTNode create a large and complex hierarchy. Writing AST by hand is verbose and unbearable (especially without an IDE).

You can do it but it is a pain.



canoo

Writing AST

```
def ast = new AstBuilder().buildFromString(
    ' println "Hello World" '
)
```

```
def ast = new AstBuilder().buildFromSpec {
    methodCall {
        variable('this')
        constant('println')
        argumentList {
            constant 'Hello World'
        }
    }
}
```

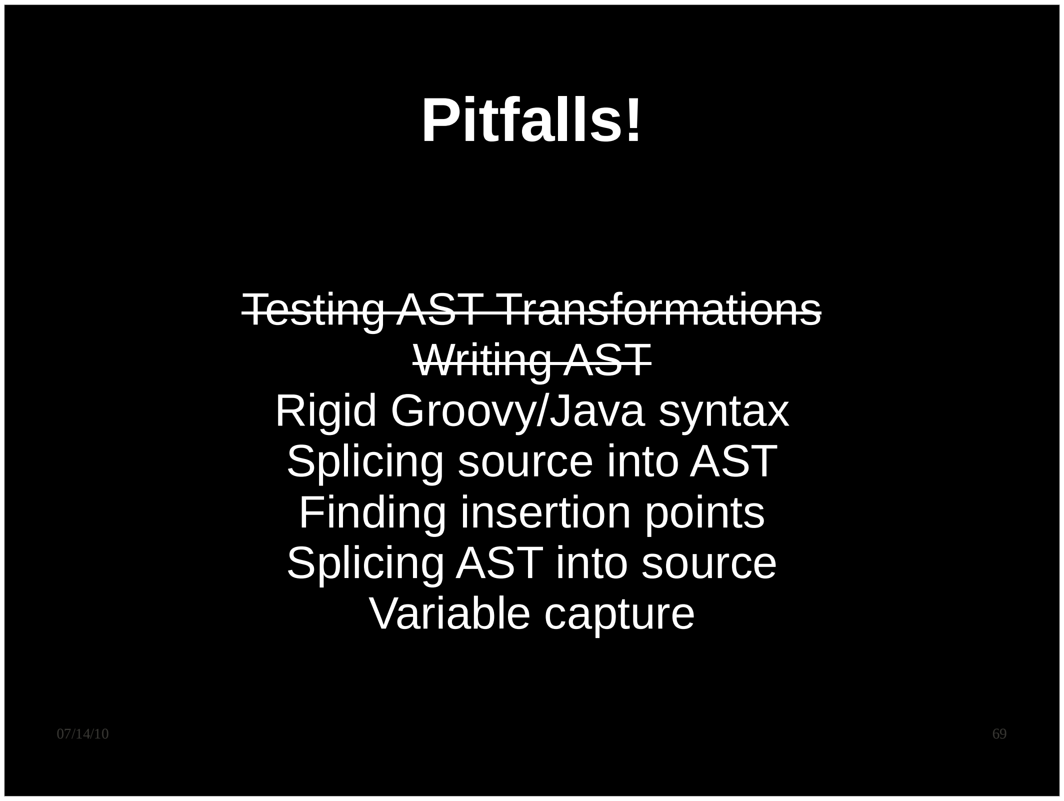
www.canoo.com **… from "Building AST Guide" on Groovy Wiki** 68

Writing AST got a lot easier in 1.7 with the introduction of AstBuilder.

The coolest version is "buildFromCode". Anything within the closure parameter is converted into AST and returned to you. Sweet.

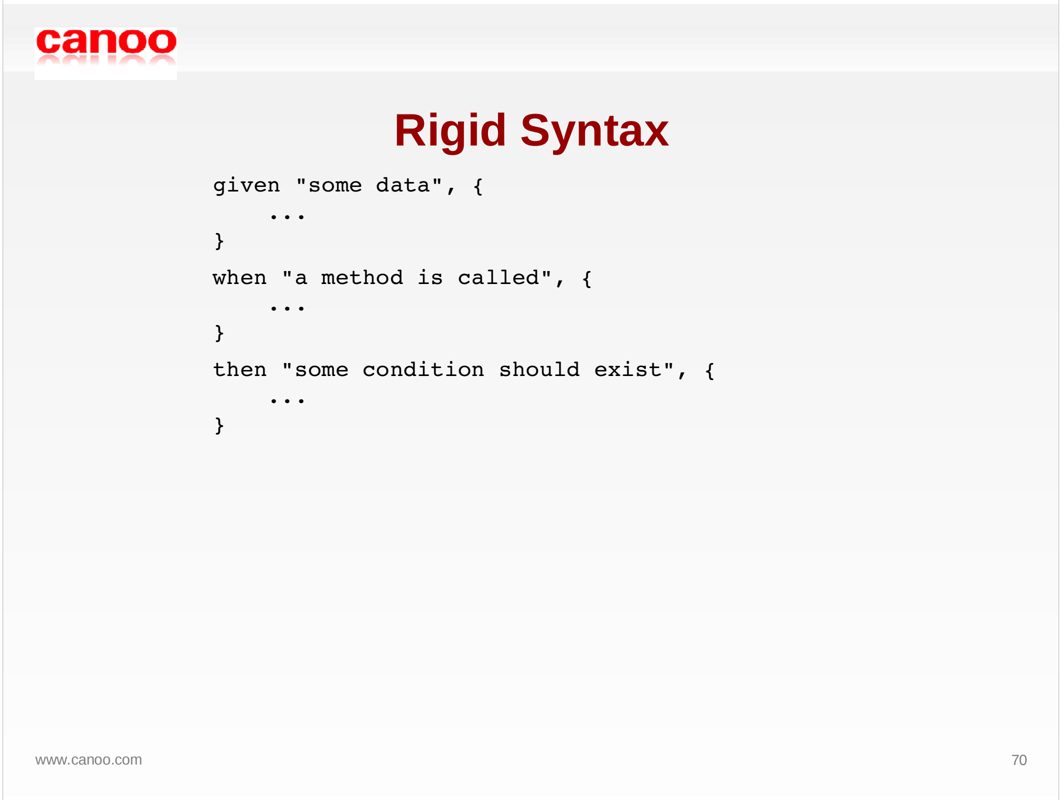
buildFromString allows easy compilation from Strings.

buildFromSpec is a DSL/Builder over the AST types. It makes it easy to embed logic into the AST creation or move code from the calling context into the new AST context. We'll see why this is so hard on the next slide.



Pitfalls!

~~Testing AST Transformations~~
~~Writing AST~~
Rigid Groovy/Java syntax
Splicing source into AST
Finding insertion points
Splicing AST into source
Variable capture


canoo

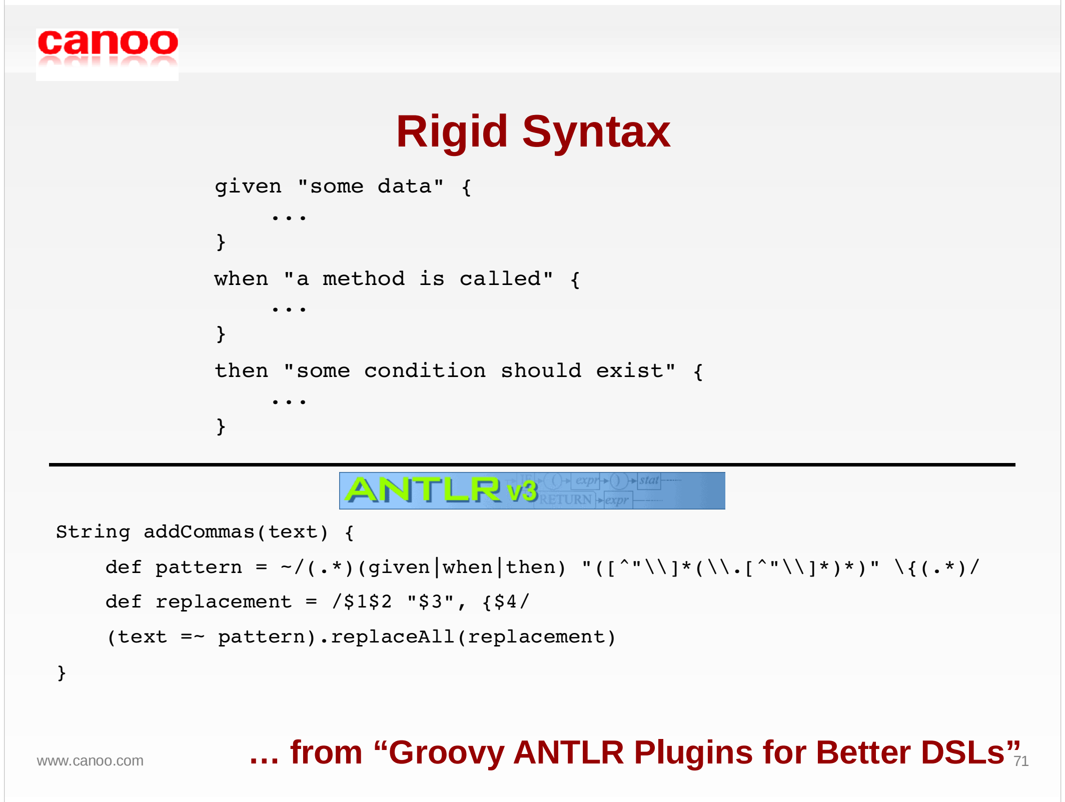
Rigid Syntax

```
given "some data", {
    ...
}
when "a method is called", {
    ...
}
then "some condition should exist", {
    ...
}
```

This is an easyb story. Easyb is a Behavior Driven Development framework with Groovy as a language.

The commas between the 1st and 2nd parameters are always required. It is part of the Groovy grammar and not easily abandoned.

OK, so commas are not the most offensive syntax, but it is a nice use case because commas *are* part of the grammar.



Rigid Syntax

```
given "some data" {
    ...
}
when "a method is called" {
    ...
}
then "some condition should exist" {
    ...
}
```

ANTLR v3

```
String addCommas(text) {
    def pattern = ~/(.*)(given|when|then) "([^"\\]*(\\.[^"\\]*)*)" \{(.*)/
    def replacement = /$1$2 "$3", {$4/
    (text =~ pattern).replaceAll(replacement)
}
```

… from “Groovy ANTLR Plugins for Better DSLs”

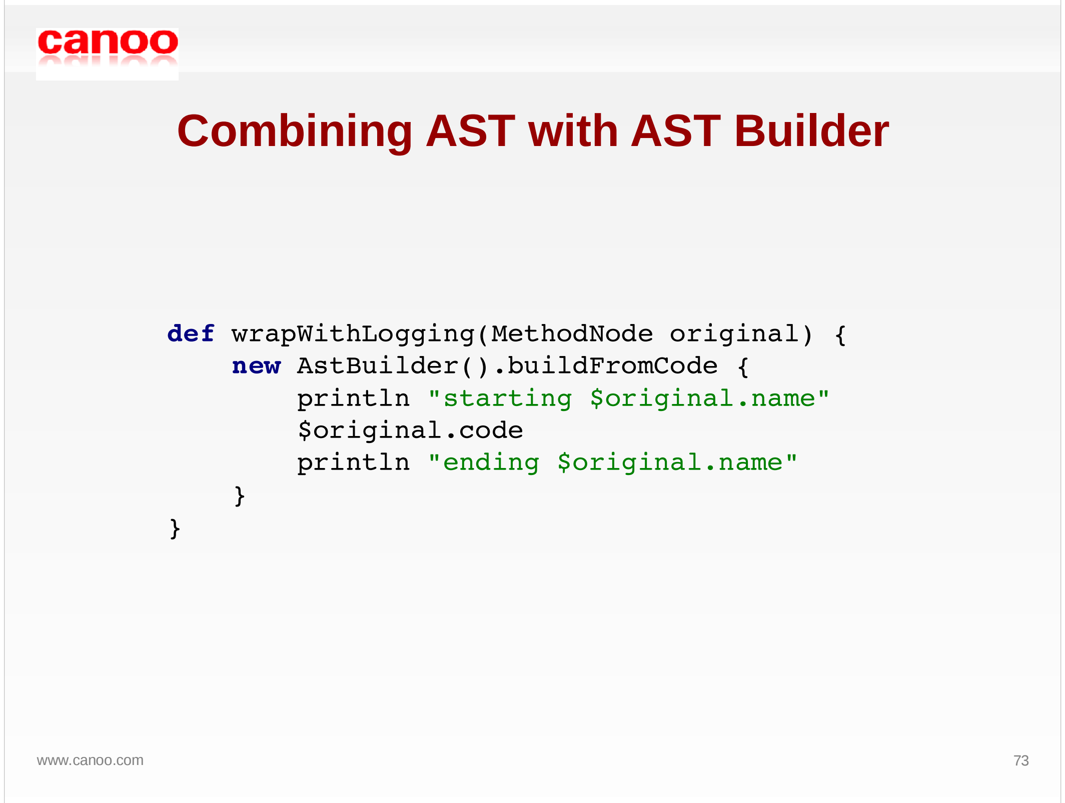
If you want to change the syntax then you can write an ANTLR plugin. This allows you to do many things, one of which is to write text based transformations over the source code, converting it into something that groovyc would recognize.

This example shows how to add the missing commas in for easyb.

Not as nice as transformations. But if you need to change the syntax then you have options.

Pitfalls!

~~Testing AST Transformations~~
~~Writing AST~~
~~Rigid Groovy/Java syntax~~
Splicing source into AST
Finding insertion points
Splicing AST into source
Variable capture



Combining AST with AST Builder

```
def wrapWithLogging(MethodNode original) {
    new AstBuilder().buildFromCode {
        println "starting $original.name"
        $original.code
        println "ending $original.name"
    }
}
```

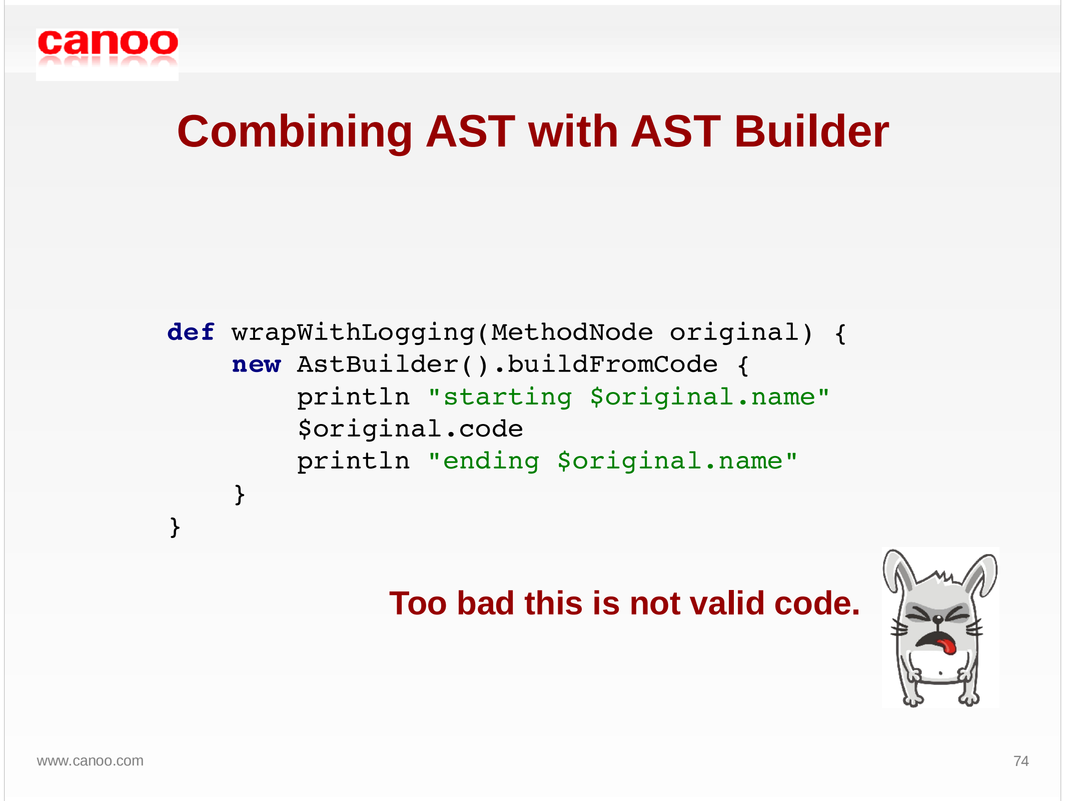
www.canoo.com 73

This example does not work in Groovy 1.7. If you want to move code from the calling context into your transformation then you can't easily use buildFromCode or buildFromString, which is the most useful of the AstBuilder API.

I would like to see this example work in Groovy 1.8, but so far my effort with GEP 4 has had a lukewarm response.

Please go read GEP 4 and let the Dev list know what you think.

If you hate it then I will quit working on it. I promise.



Combining AST with AST Builder

```
def wrapWithLogging(MethodNode original) {
    new AstBuilder().buildFromCode {
        println "starting $original.name"
        $original.code
        println "ending $original.name"
    }
}
```

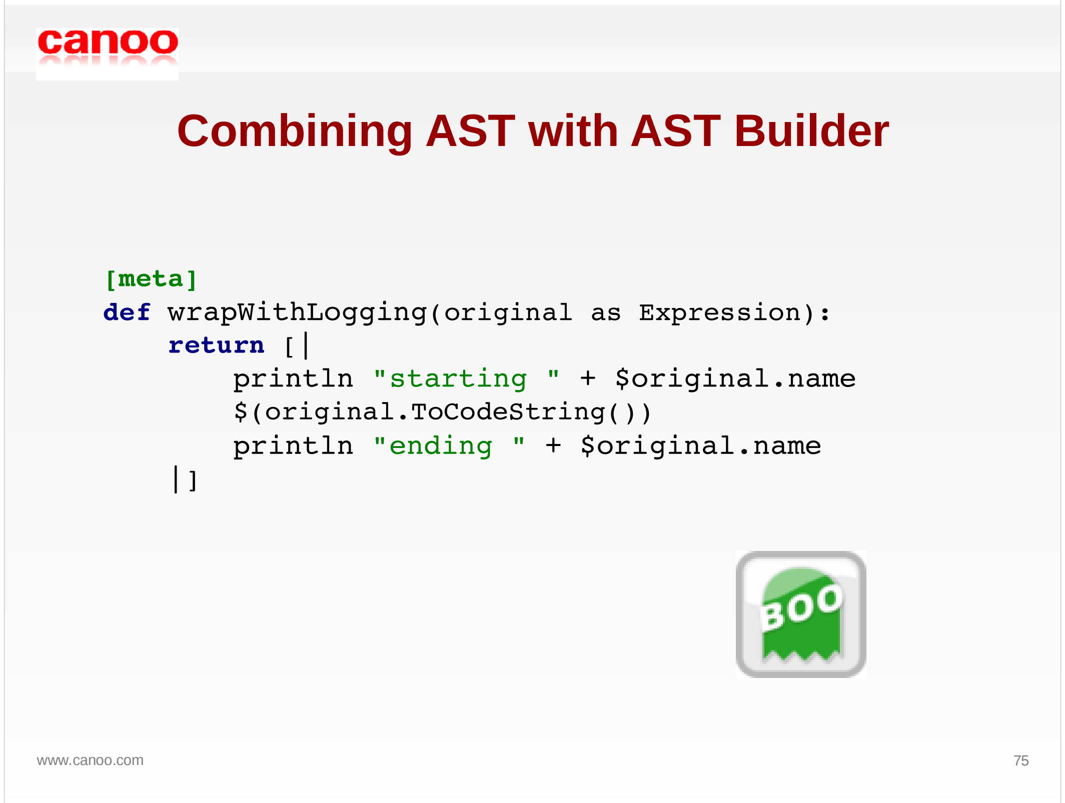
Too bad this is not valid code.

This example does not work in Groovy 1.7. If you want to move code from the calling context into your transformation then you can't easily use buildFromCode or buildFromString, which is the most useful of the AstBuilder API.

I would like to see this example work in Groovy 1.8, but so far my effort with GEP 4 has had a lukewarm response.

Please go read GEP 4 and let the Dev list know what you think.

If you hate it then I will quit working on it. I promise.



canoo

Combining AST with AST Builder

```
[meta]
def wrapWithLogging(original as Expression):
    return [|
        println "starting " + $original.name
        $(original.ToCodeString())
        println "ending " + $original.name
    |]
```

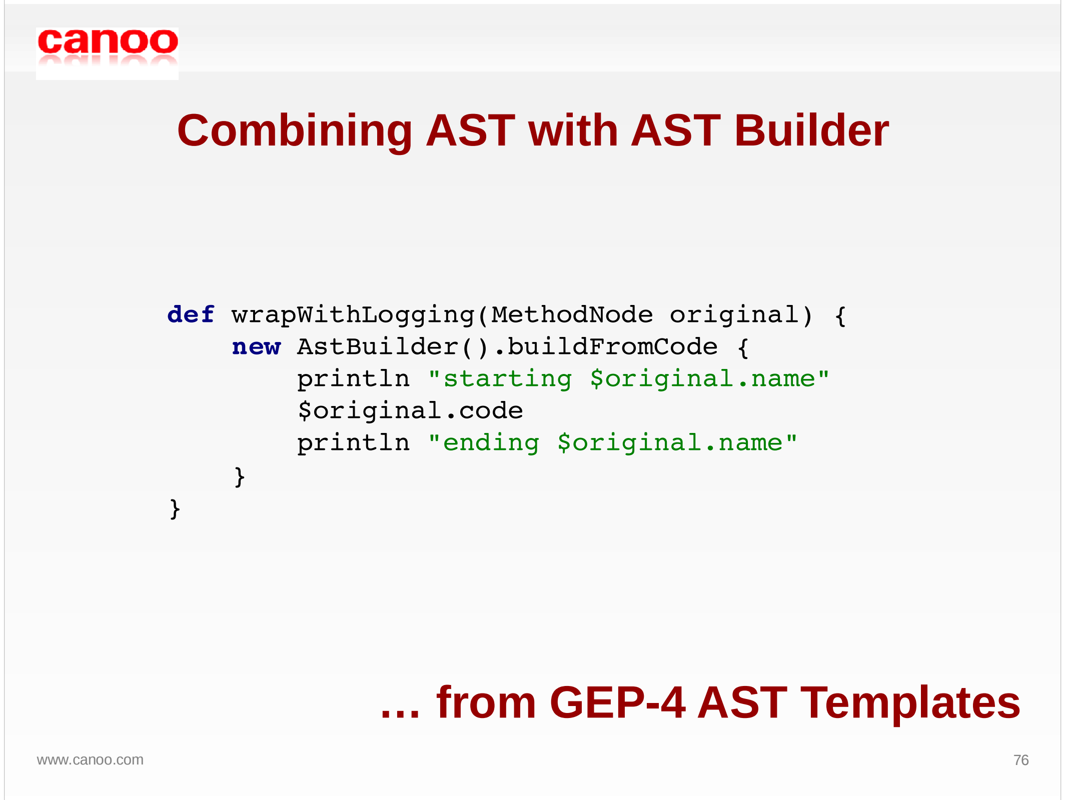
Boo has a very nice compiler metaprogramming framework, and I would like to see Groovy's evolve to match it.

Instead of “ASTTransformation” subclasses and annotations, a compile time method is simply marked as [meta]

Instead of wrapping code in AstBuilder, code is transformed by wrapping it in [| |].

Code from the calling context is spliced into an AST build using the $ syntax.

Problem: The compiler must know to dispatch a method call to a [meta] method at compile time. This is not dynamic, runtime dispatch. It is static dispatch that requires type information. This would further complicate the method dispatch of Groovy, which is already complicated.



Combining AST with AST Builder

```
def wrapWithLogging(MethodNode original) {
    new AstBuilder().buildFromCode {
        println "starting $original.name"
        $original.code
        println "ending $original.name"
    }
}
```

… from GEP-4 AST Templates

This example does not work in Groovy 1.7. If you want to move code from the calling context into your transformation then you can't easily use buildFromCode or buildFromString, which is the most useful of the AstBuilder API.

I would like to see this example work in Groovy 1.8, but so far my effort with GEP 4 has had a lukewarm response.

Please go read GEP 4 and let the Dev list know what you think.

If you hate it then I will quit working on it. I promise.

Pitfalls!

~~Testing AST Transformations~~
~~Writing AST~~
~~Rigid Groovy/Java syntax~~
~~Splicing source into AST~~ GEP-4 in 1.8?
Finding insertion points
Splicing AST into source
Variable capture

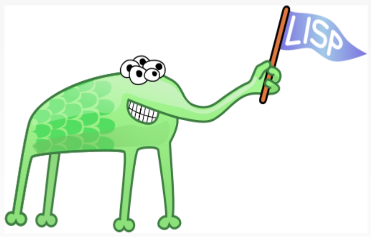
Finding Insertion Points
&
Splicing AST into Source

```
public interface ASTTransformation {
    public void visit(ASTNode[] nodes,
                            SourceUnit source);
}
```

Finding Insertion Points & Splicing AST into Source

```
def x = "some value"
setNull x
assert x == null
```

Warning... Lisp Ahead



Finding Insertion Points
&
Splicing AST into Source

```
def x = "some value"
setNull x
assert x == null
```

```
(defmacro setNull (var)
    (list ’setq var nil))
```

Variable Capture

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

Groovy Code Generation

Good Things

- When nothing else will do
- To call functions without evaluating arguments
- To modify variables in calling scope

Bad Things

- Difficult to write
- Difficult to write *correctly*
- Source code clarity
- Runtime clarity
- Version compatibility

canoo

Groovy Code Generation

Good Things

- When nothing else will do
- To call functions without evaluating arguments
- To modify variables in calling scope

Bad Things

- Difficult to write
- Difficult to write *correctly*
- Source code clarity
- Runtime clarity
- Version compatibility

… from "On Lisp" by Paul Graham

Q. You are marooned on a deserted island with only one programming language. Which do you want?

Q. You are marooned on a deserted island with only one programming language. Which do you want?

A. One with AST Transformations

...and a freakin' sweet logo

Thanks!

- What to do next:
  - Groovy Wiki and Mailing List is amazingly helpful
  - Use your creativity and *patience*
  - Come to Hackergarten at Canoo
  - http://hamletdarcy.blogspot.com & @HamletDRC

- I am speaking at:
  - CZ Jug
  - SpringOne/2GX
  - JavaOne
  - Your JUG? Please?