Static type checking & compilation
in Groovy 2

Cédric Champeau

VMware

About me

- Past : Groovy contributor
  - Bugfixes
  - Modules: @Bytecode AST xform, GFreeMarker
  - Core: compilation customizers, @xInterrupt, ...
  - Used Groovy as a DSL for natural language processing
- Present: Core Groovy committer
  - Working on bugfixes but main focus on **Groovy 2.0**
    - Static type checking
    - Static compilation
- Follow me
  - Twitter: @CedricChampeau
  - Google+: http://gplus.to/cchampeau
  - Blog: http://jroller.com/melix

Groovy 2

- Modularity
  - several jars for modules (swing, ant, xml, json, ...)
  - groovy.jar now 3MB
  - extension modules
- JDK7 enhancements
  - project coin
  - invoke dynamic support (use -indy jars)
- Static theme
  - Static type checking
  - Static compilation
- 2.0.1 has just been released!
  - get it while it's hot!

Static Type Checking

- Goal
  - Find errors at **compile time (fail early)**
  - because lots of code do not use dynamic features of Groovy
  - so many bugs can be discovered before production
  - Make java developers even happier
- Turn the compiler **grumpy**
  - Report typos
    - missing method/property
  - Extension methods (aka DefaultGroovyMethods)
  - Type check assignments
  - Perform type inference
    - method/closure return type inference
    - generics type inference

Static Type Checking

It's **optional**

- Triggered using @TypeChecked annotation
- Annotate a class or a method

```
@groovy.transform.TypeChecked
class Example1 {
    void foo() {}
    void bar() {}

    void method() {
        foo() // correct
        baz() // cannot find matching method Example1#baz()
    }
}
```

Fails at compile time, not runtime!

Static Type Checking basics

```
@groovy.transform.TypeChecked
class Example2 {
    String name = 'Cedric'

    void method() {
        println naame // typo, the variable [naame] is undeclared
    }
}
```

```
@groovy.transform.TypeChecked
class Example3 {
    String name = 'Cedric'

    void method() {
        println "My name is $naame" // in gstrings too, the variable [naame] is undeclared
    }
}
```

```
@groovy.transform.TypeChecked
class Example4 {
    void method() {
        int x = 0
        x += new Object() // Cannot find matching method int#plus(Object)
    }
}
```

STC basics (cont'd)

```
@groovy.transform.TypeChecked
class Example6 {
    void method() {
        Set set = 'Bar' // Cannot assign a value of type String to variable of type Set
    }
}
```

```
@groovy.transform.TypeChecked
class Example5 {
    void method() {
        Date[] arr = new Date[2]
        arr[0] = new Date() // ok
        arr[1] = 'Hello' // Cannot assign value of type String to variable of type Date
    }
}
```

```
@groovy.transform.TypeChecked
class Example7 {
    int method() {
        'foo' // cannot return value of type String on method returning type int
    }
}
```

STC basics (cont'd)

```
@groovy.transform.TypeChecked
class Example9 {
    boolean test() { true }
    void m() {
        int x = 1 + test() // return type checking
    }
}
```

```
@groovy.transform.TypeChecked
class Example10 {
    String test() { new StringBuilder('transparent conversion to string') }
    void m() {
        println "${test()}"
    }
}
```

```
@groovy.transform.TypeChecked
class Example11 {
    int test(String val) {
        val.toInteger() // DGM methods
    }
}
```

@TypeChecked options

- if set on a class
  - type checks the whole class
  - type checks inner classes
  - type checks closures defined within the class
- if set on a method
  - type checks only that method
  - type checks closures defined within the method body
- you can prevent some code from being type checked

```
import groovy.transform.TypeChecked
import static groovy.transform.TypeCheckingMode.*

@TypeChecked(SKIP)
int foo() { 'foo' }
```

STC advanced features

instanceOf without needing explicit cast

```
@groovy.transform.TypeChecked
class Example12 {
    void test(Object val) {
        if (val instanceof String) {
            println val.toUpperCase() // no need for explicit cast
        } else if (val instanceof Number) {
            println ("X"*val.intValue()) // String#multiply(int) from DGM
        }
    }
}
```

Type inference

```
@groovy.transform.TypeChecked
clas @groovy.transform.TypeChecked
    class Example15 {
        void test() {                                                    ce
            List<? extends Comparable> list = ['a',123] // LUB inferrence
        }                                                                List<Number>
    }
}
```

LUB = Lowest Upper Bound

represents lowest "super" type that classes have in common

May be "virtual"

STC advanced features (cont'd)

Compatible with "Groovy way" of instantiating objects

```
Dimension d = [100, 200]
```

Closure return type inference

```
def cl = { new Date() }
long time = cl().time
```

Closure type arguments

```
def closure = { int x, int y -> x+y }
closure(1,2)
closure('1','2') // complains
```

Flow Typing

Use type inference to remove the need for explicit casts

```
@groovy.transform.TypeChecked
class FlowTyping2 {
    void main(String... args) {
        def var = 123 // inferred type is an int
        int x = var // var is an int, no problem
        var = "123" // assign var with a String
        x = var.toInteger() // no problem and no need to cast
        var = 123
        x = var.toUpperCase() // error because var is of type int at this point
    }
}
```

Future options

- annotation at package level
  - probably not (especially when you think about precompiled classes)
- custom file extension
  - more likely
- compiler option
  - most probably
  - but not specific to @TypeChecked
  - generic way of applying AST transforms transparently
    - think of *GroovyShell* compilation customizers but for *groovyc*

see *this thread on the dev mailing list for further details*

Static Type Checking gotchas

Example

This won't pass static type checking

```
@TypeChecked
class WontPass {
    void run() {
        Integer.metaClass.foo = { println 'foo' }
        123.foo()
    }
}
```

metaclass is unsupported

Method won't be recognized

Metaclasses can be changed from any thread

- even if you change the metaclass in a not statically checked method
- the changes won't be "visible" to a static type checker

Closure parameter types

Parameters of closure require explicit type information

```
@groovy.transform.TypeChecked
class Example16 {
    void test() {
        ['a','b','c'].collect { it.toUpperCase() }
    }
}
```

type checker cannot determine type of *it*

add explicit type information

```
@groovy.transform.TypeChecked
class Example16Fixed {
    void test() {
        ['a','b','c'].collect { String it -> it.toUpperCase() }
    }
}
```

Enhancement proposal requiring changes to Groovy Core API is available

Closure parameter types (bonus)

- Exercise
  - Compare DefaultGroovyMethods#collect
  - with DefaultGroovyMethods#times

Closure shared variables

Tracking assignments of closure shared variables

The type checker enforces "good practice"

ensure that method calls are only possible on the LUB of all assignments

```
@groovy.transform.TypeChecked
class ClosureSharedVariable {
    void test() {
        def var = "String"
        def cl = { var = new Date() }
        cl()
        var = var.toUpperCase()
    }
}
```

var is assigned inside a closure :
it's a *closure shared variable*

in general, we cannot ensure when
a closure gets executed

[Static type checking] - A closure shared variable [var] has been assigned with various types and the method [toUpperCase()] does not exist in the lowest upper bound of those types: [java.lang.Comparable]. In general, this is a bad practice (variable reuse) because the compiler cannot determine safely what is the type of the variable at the moment of the call in a multithreaded context.

Closure shared variables (2)

But this would work fine

```
class A { void foo() {} }
class B extends A { void bar() {} }

@groovy.transform.TypeChecked
class ClosureSharedVariable {
    void test() {
        def var = new A()
        def cl = { var = new B() }
        cl()
        var.foo() // var is at least an instance of A
    }
}
```

Closures

- Closures at type checked using the "default" configuration
  - meaning the default delegation strategy (owner first)
  - you cannot change the owner
  - you cannot change the delegate
- Nothing prevents you from changing them
  - but type checker will be **wrong**

Method selection gotchas

What's wrong with this (if it's statically checked) ?

```
static String foo(String s) {
    'String'
}
static String foo(Integer s) {
    'Integer'
}
static String foo(Boolean s) {
    'Boolean'
}
['foo',123,true].each {
    if (it instanceof String) {
        foo(it)
    } else if (it instanceof Boolean) {
        foo(it)
    } else if (it instanceof Integer) {
        foo(it)
    }
}
```

#foo(Object)

instanceof removes need for cast

How does STC works?

- It's an AST transformation!
  - Technically, it doesn't really transform anything
  - but it makes intensive use of node metadata
    - see ASTNode#getNodeMetaData
  - runs at *instruction selection* phase (late)
- Some compiler framework changes were needed to support
  - regular compilation
  - invokedynamic (1.7+ JVMs)
  - static compilation

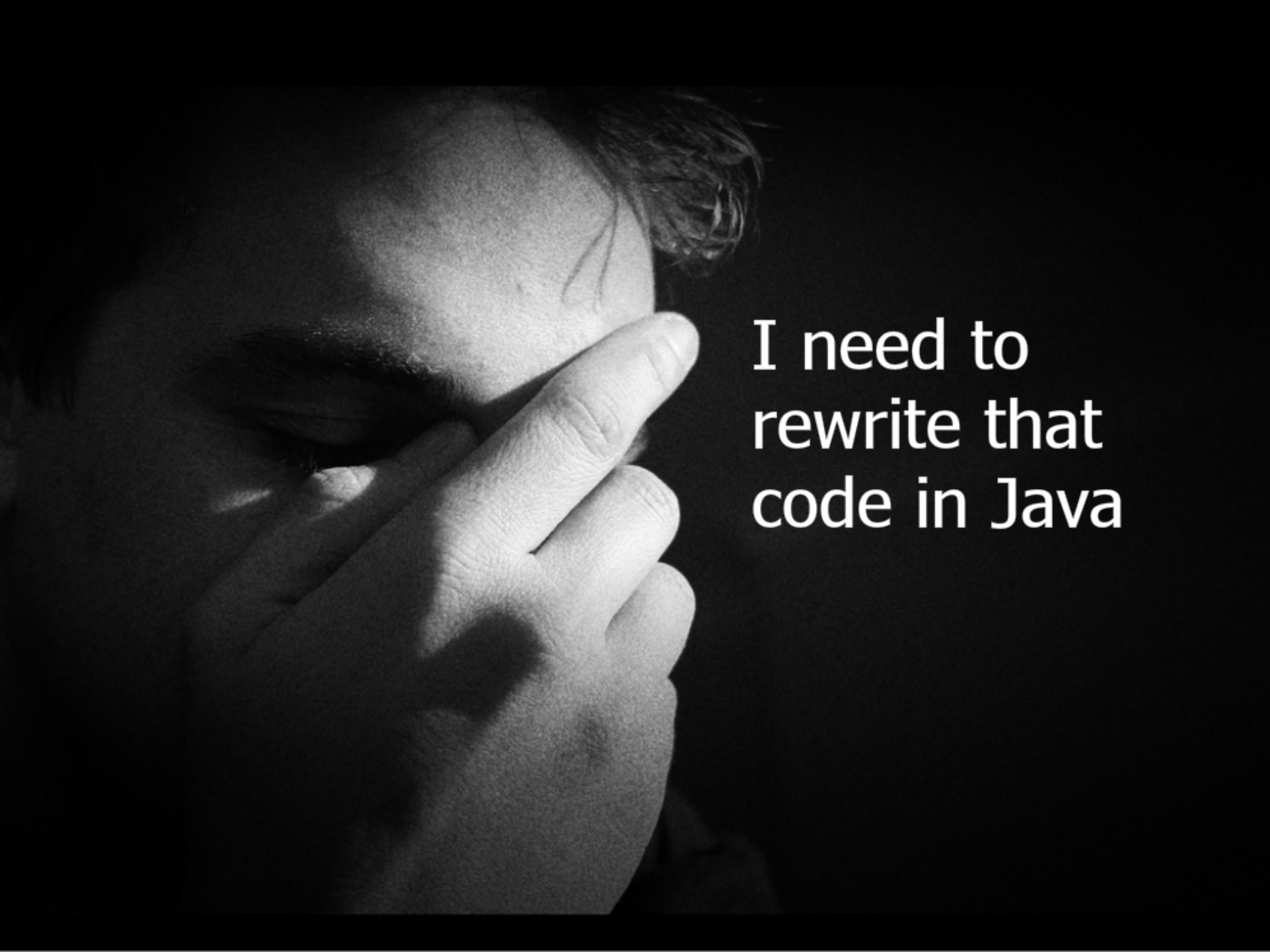
Testing the type checker

- It's an AST transform
- We want to test the result of the transform
- @ASTTest to the rescue
  - can be used for **any** AST transformation

```
import groovy.transform.ASTTest
import static org.codehaus.groovy.control.CompilePhase.*
import static org.codehaus.groovy.transform.stc.StaticTypesMarker.*
import static org.codehaus.groovy.ast.ClassHelper.*

int n = 10
for (int i in 1..n) {
    @ASTTest(phase=INSTRUCTION_SELECTION, value= {
        assert node.getNodeMetaData(DECLARATION_INFERRED_TYPE) == int_TYPE
    })
    def j = i
}
```

Static compilation

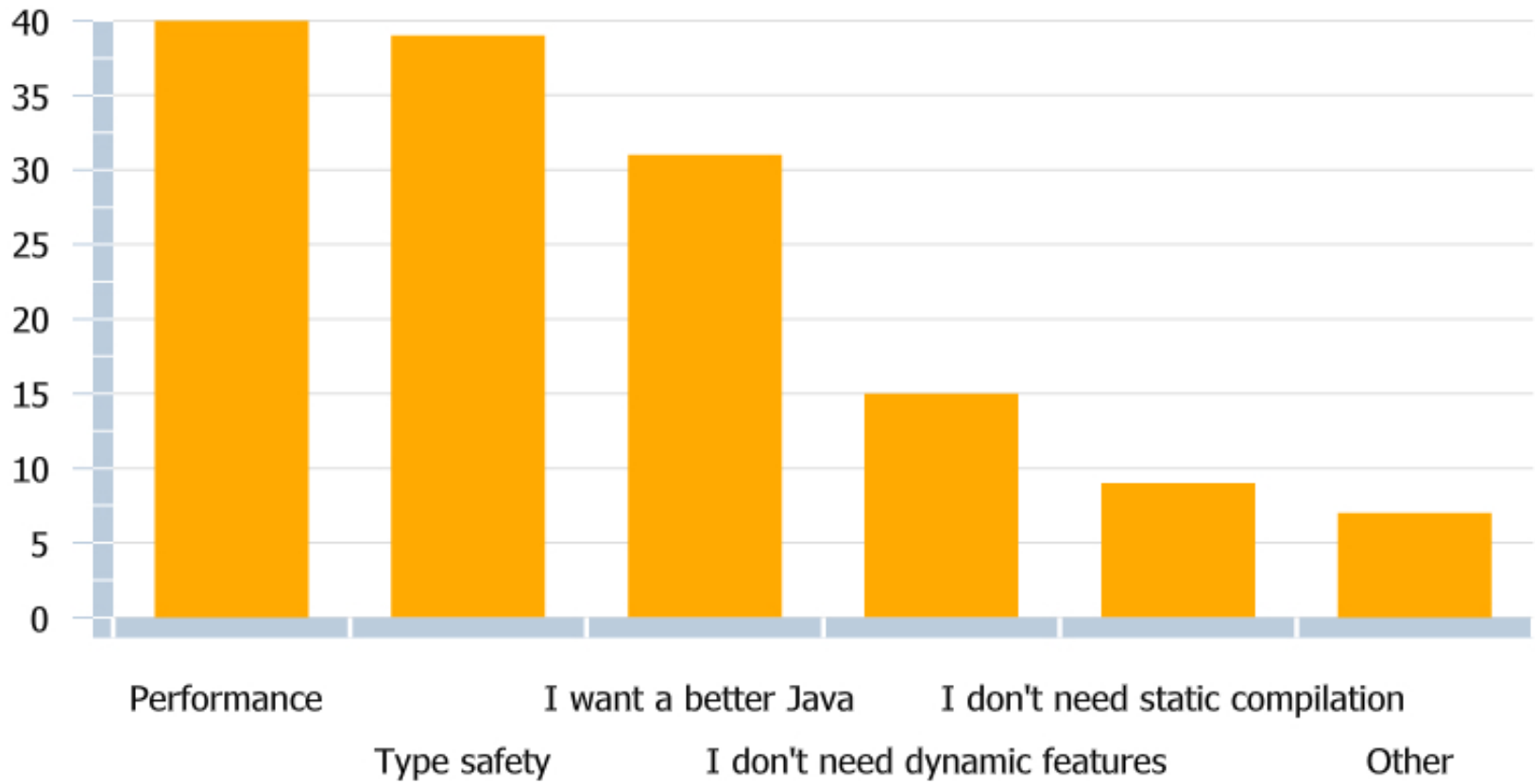

I need to
rewrite that
code in Java

My manager wants me to do it in VB

Why do you want static compilation?

Results from a previous poll (early talk about static compilation)

8% of viewers voted



Why static compilation?

- If you perform static type checking...
  - you cannot ensure dynamic semantics
  - so why can't you statically compile ?
- Benefits
  - Static type checking (type safety)
  - Improved performance
    - not everyone has JDK7+ (InvokeDynamic)
  - Bytecode size (work on progress)
  - Immunity to monkey patching (useful for framework developers)
- But you loose...
  - Dynamic features (categories, metaclasses, ...)
  - Dynamic method dispatch
  - Dynamic Groovy semantics (part of)

In Groovy 2.0

- Static compilation is **optional**
- Relies on the *@TypeChecked* AST transformation
- Built to be as close as possible as dynamic Groovy semantics
  - but there are important differences
- Unlike Groovy++, it's just static compilation
  - no added features
  - no grammar changes
- Features and design choices discussed with the community
- Work still in progress

  bytecode optimizations

  bugfixes

  we need your pull requests!

Static method dispatch

What does the following Groovy program outputs ?

```
def foo(String a) { 'STRING' }
def foo(Date a) { 'DATE' }

Object o = new Date()
foo(o)
```

target method is chosen at *runtime*

In Java, would fail at compile time

In static Groovy (@CompileStatic), method is chosen at compile time

- we know that when foo() is called, o must be a Date
- so foo(Date) is chosen
- just like dynamic Groovy
- that is called inference based dispatch

Static method dispatch (cont'd)

But be careful !

```
@CompileStatic
class StaticDispatch {
    static def foo(Object a) { 'OBJECT' }
    static def foo(String a) { 'STRING' }
    static def foo(Date a) { 'DATE' }

    static def bar(Object o) {
        foo(o)
    }

    static void main(String... args) {
        println foo('string')
        println bar('string')
    }
}
```

Inferred type of *o* is always *Object*

will output *OBJECT* !

Bytecode!

- Fibonacci example
  - not the smartest way to do this, yet perfectly suitable for micro-benchmarks
  - Java

```
public class Fib {
    public int fib(int n) {
        return n<2?n:fib(n-1)+fib(n-2);
    }
}
```

  - Groovy

```
int fib(int n) {
    n<2?n:fib(n-1)+fib(n-2)
}
```

Bytecode! (cont'd)

Java

```
public fib(I)I
  L0
   ILOAD 1
   ICONST_2
   IF_ICMPGE L1
   ILOAD 1
   GOTO L2
  L1
   ALOAD 0
   ILOAD 1
   ICONST_1
   ISUB
   INVOKEVIRTUAL Fib.fib (I)I
   ALOAD 0
   ILOAD 1
   ICONST_2
   ISUB
   INVOKEVIRTUAL Fib.fib (I)I
   IADD
  L2
   IRETURN
```

Groovy 1.8 (primopts)

```
public fib(I)I
 L0
  INVOKESTATIC script1339060140068.$getCallSiteArray ()[L
  ASTORE 2
  INVOKESTATIC org/codehaus/groovy/runtime/BytecodeInterf
  IFEQ L1
  INVOKESTATIC org/codehaus/groovy/runtime/BytecodeInterf
  IFEQ L1
  GETSTATIC script1339060140068.__$stMC : Z
  IFNE L1
  INVOKESTATIC org/codehaus/groovy/runtime/BytecodeInterf
  IFNE L1
  GOTO L2
 L1
  ILOAD 1
  ICONST_2
  IF_ICMPGE L3
  ICONST_1
  GOTO L4
 L3
  ICONST_0
 L4
  IFEQ L5
  ILOAD 1
  GOTO L6
 L5
  ALOAD 2
  LDC 1
  AALOAD
  ALOAD 2
  LDC 2
  AALOAD
```

Bytecode! (cont'd)

Java

```
public fib(I)I
  L0
   ILOAD 1
   ICONST_2
   IF_ICMPGE L1
   ILOAD 1
   GOTO L2
  L1
   ALOAD 0
   ILOAD 1
   ICONST_1
   ISUB
   INVOKEVIRTUAL Fib.fib (I)I
   ALOAD 0
   ILOAD 1
   ICONST_2
   ISUB
   INVOKEVIRTUAL Fib.fib (I)I
   IADD
  L2
   IRETURN
```

Groovy 2.0 (@CompileStatic)

```
public fib(I)I
 L0
  ILOAD 1
  ICONST_2
  IF_ICMPGE L1
  ICONST_1
  GOTO L2
 L1
  ICONST_0
 L2
  IFEQ L3
  ILOAD 1
  GOTO L4
 L3
  ALOAD 0
  ILOAD 1
  ICONST_1
  ISUB
  INVOKEVIRTUAL script1339060545440.fib (I)I
  ALOAD 0
  ILOAD 1
  ICONST_2
  ISUB
  INVOKEVIRTUAL script1339060545440.fib (I)I
  IADD
 L4
  IRETURN
```

Performance

	Fib	Pi	Binary trees
1.8	14.22s	4.76s	28.9s
1.8 (primopts)	1.78s	0.53s	13.1s
2.0.0 rc3 (indy)	9.40s	2.00s	29.98s
2.0.0 rc3 (@CS)	0.79s	0.56s	2.6s
java	0.80s	0.57s	1.58s

Questions?

vmware

Thank you!

THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
TASHAKKUR ATU
YAQHANYELAY
SUKSAMA
EKHMET
MEHRBANI
GRAZIE
MAAKE
KOMAPSUMNIDA
GOZAIMASHITA
EFCHARISTO
JUSPAXAR
PALDIES
BOLZÏN
MERCI
BÏYAN
SHUKRIA
TINGKI
SPASIBO
SNACHALHUYA
NUHUN
CHALTU
WABEEJA
MAITEKA
DHANYADAAD
ATTO
MIERSI
DENKAUJA
NENACHALHYA
UNALCHEESH
GUI
HATUR
EKOJU
SIKOMO
MAKETU
MINMONCHAR
MERASTAWHY
GAEJTHO
SAHCO
LAH
TAVYAPUCH
MEDAWAGSE
AGUYJE
FAKAAUE
BADIKA