

■ ■ ■ GROOVY



Mischa Kölliker
Guido Schmutz

Zürich, 24.06.2008

trivadis
makes IT easier. ■ ■ ■

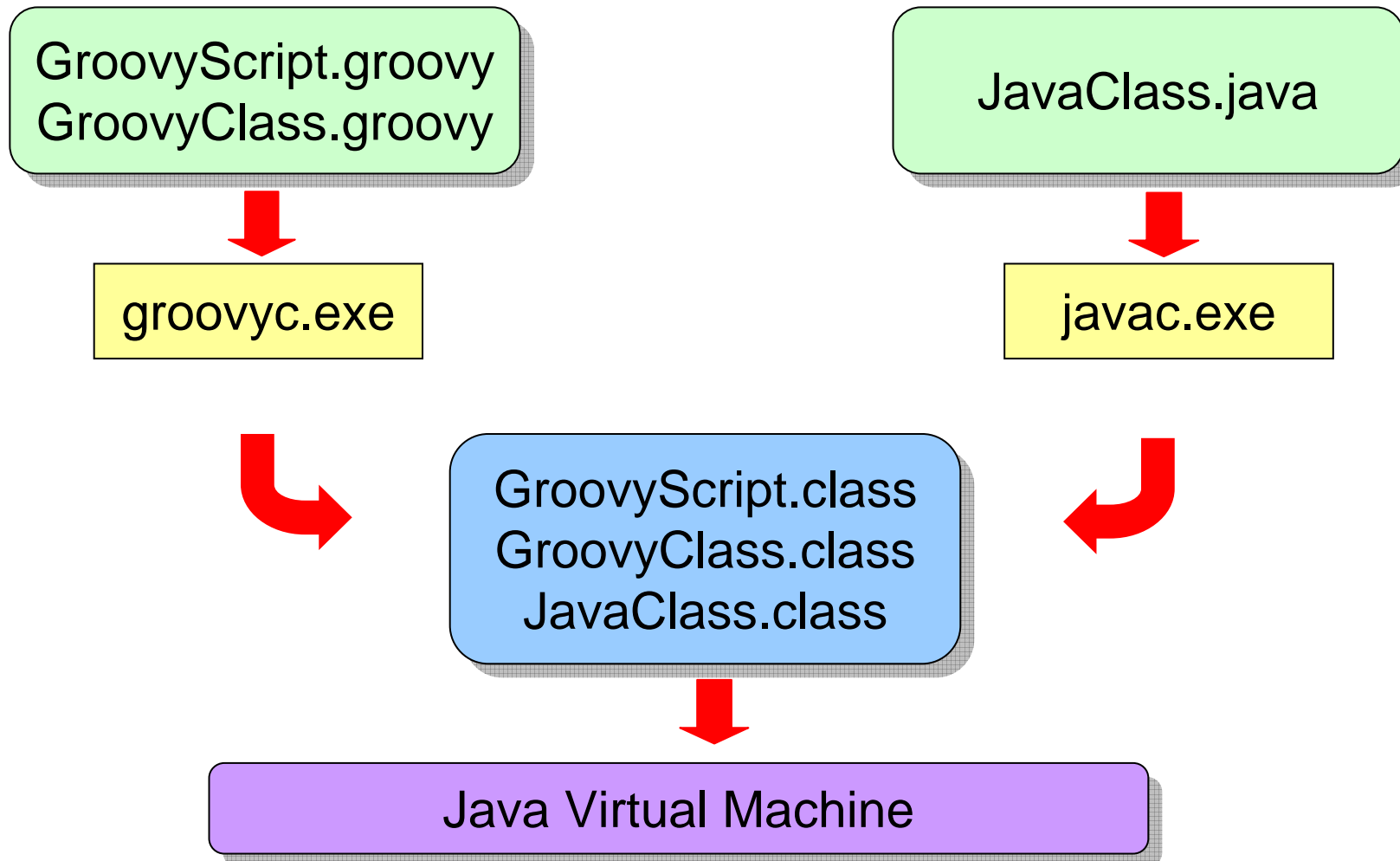
Agenda



Data are always part of the game.

- Groovy: Java on dope
- Builders
- Groovy on Windows
- What about performance?
- Wrap up

Groovy.compareTo(Java)



Groovy.compareTo(Java)



Groovy

- compiled
- dynamic typed
- weak typed
- "execute" Source

Java

- compiled
- static typed
- weak typed
- execute compiled class only

Groovy basics



- Implicit classes

```
println 'Hello Groovy'
```

- Java:

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

Groovy Collections



```
list = 1..10           // init with 1 - 10
list += "sometext"    // add to the collection
list -= 5              // remove element '5' from the collection

list.each { println it } // iterate over the collection

// compare collections
assert list == [1,2,3,4,6,7,8,9,10,'sometext']

listtwo = [1,555,'text',4, new Date()]

listtwo[20] = 8 // dynamically extend
```

Groovy Maps



```
def map = [:] // empty map

// init a map
map = [first:'text1', second:'text2', third:'text3']

// access via property notation
println map.first

// iterate
map.each { println "$it.key -> $it.value" }
```

GPath



- Java:

```
Method[] methods = this.getClass().getMethods();
Pattern pattern = Pattern.compile("get.*");
List result = new ArrayList();
for (int i = 0; i < methods.length; i++) {
    String name = methods[i].getName();
    if (pattern.matcher(name).matches())
        result.add(name);
}
Collections.sort(result);
```

GPath - Groovy



```
this.class.methods.name.grep(~/get.*/).sort()
```

Closures



- Methods without a name
- Internally implemented as an inner class with one method
- Often, implicit variables like ,it' are available
- The last expression will be automatically returned
 - This is valid for all methods

```
def add = { x,y -> x + y } // define closure
```

```
println add(25,26) // call it
```

Closures with iterators



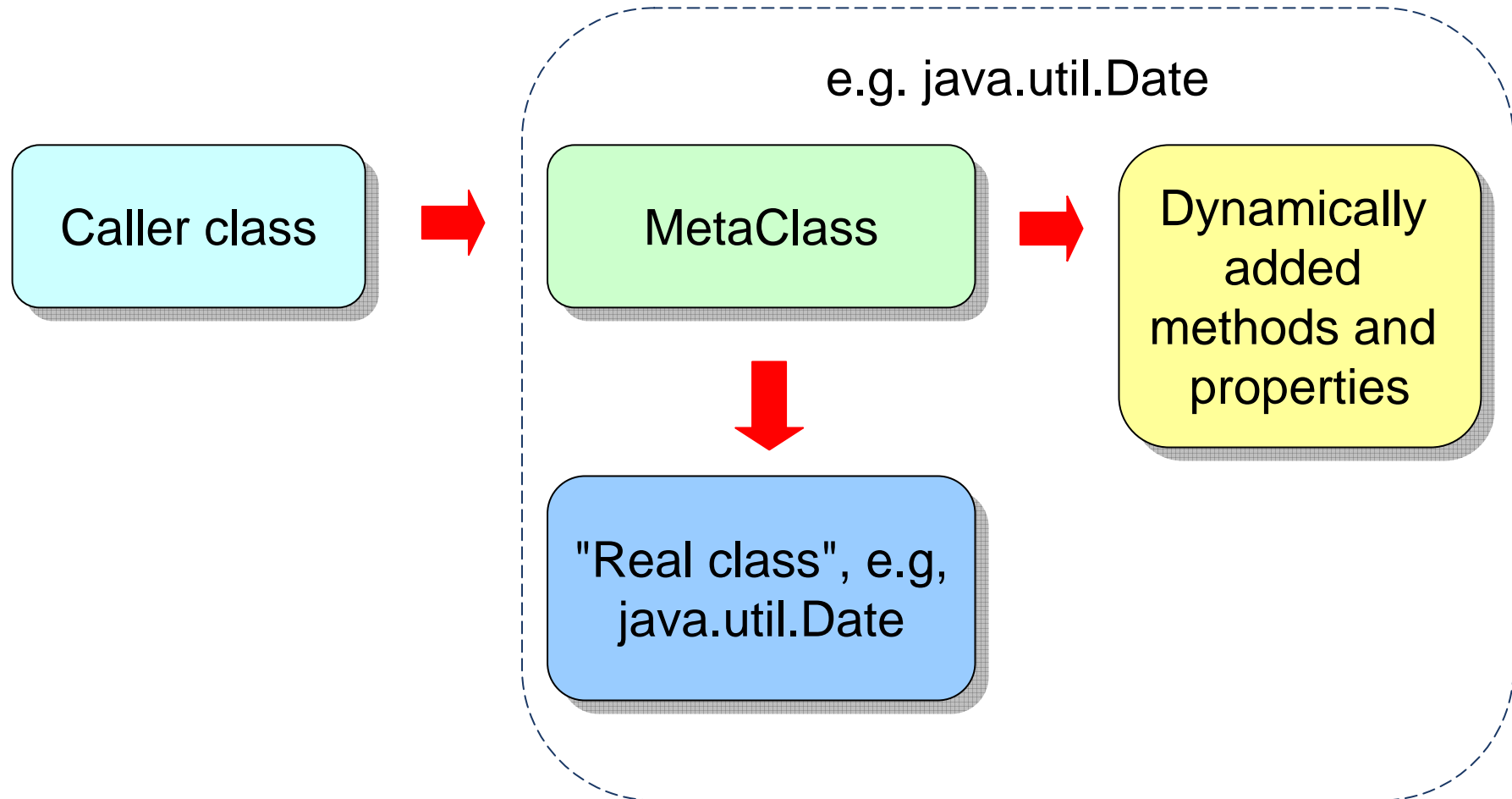
- Work like the body of a ,for‘ or ,while‘ loop

```
10.times { x -> print x } // with explicit parameter
10.times { print it } // with implicit parameter 'it'

map.each { println "$it.key -> $it.value" }
```

- Beyond `times` and `each`, there is `reverseEach`, `eachMatch`, `find`, `findAll`, `findIndexOf`, `any`, `every`, `grep`, `collect`, `collectAll`, `groupBy`, `inject`, `sum`, `min`, `max`, `upTo`, `downTo`, `step` and more iterating methods which take a closure
- Look at `org.codehaus.groovy.runtime.DefaultGroovyMethods` to find out more

The Meta Object Protocol (MOP)



The MOP: intercept, relay, pretend



- Meta Object Protocol
- The heart of Groovy
- „Simulate“ properties:

```
class MopProps
{
    def map

    Object getProperty(String property)
    {
        map ? map[property] : null
    }
}

void setProperty(String property, Object newValue)
{
    if (!map)
        map = [:]

    map[property] = newValue
}
}
```

Pretend properties: usage



```
MopProps props = new MopProps()
props.cat           = 'miau'
props.'spacy name' = 'somevalue'
```

```
println props.cat
println props.'spacy name'
println props.unknownprop
```

```
class MopProps
{
    def map

    Object getProperty(String property)
    {
        map ? map[property] : null
    }

    void setProperty(String property, Object newValue)
    {
        if (!map)
            map = [:]

        map[property] = newValue
    }
}
```

- There are also `get()` and `set()`, only for unknown properties

Agenda



Data are always part of the game.

- Groovy: Java on dope
- Builders
- Groovy on Windows
- What about performance?
- Wrap up

Groovy Builders



```
mb = new MarkupBuilder()
mb.html {
    head {
        title 'This is my title'
    }
    body {
        p 'This is my paragraph.'
        completelyUnknownElement(miau: 'quak')

        table {
            (1..2).each { x ->
                tr {
                    (1..3).each {
                        td (x + '-' + it)
                    }
                }
            }
        }
    }
}
```

Groovy Builders



- Builders are a set of nested Closures which
 - Pretend available methods
 - Delegate method calls to the builder
- Builders are executable configuration
- Builders produce their output (if any) immediately
 - Not like config files, which are first parsed and then queried
 - Also not like a template which can be reused

Agenda



Data are always part of the game.

- Groovy: Java on dope
- Builders
- Groovy on Windows
- What about performance?
- Wrap up

Scriptom: Groovy on Windows



```
'cmd /c notepad'.execute()  
sleep(100)
```

```
Scriptom.inApartment // better use that to automatically clean up resources  
{  
    // (ActiveX objects etc.)  
    def wshell = new ActiveXObject('WScript.Shell')  
  
    wshell.AppActivate('Untitled - Notepad')  
    wshell.sendKeys('First name: Mischa{ENTER}')  
    wshell.sendKeys('Last name: Köllker{ENTER}')  
  
    key = /HKCU\software\Microsoft\Windows\CurrentVersion\Internet Settings\User Agent/  
    println String.valueOf(wshell.regRead(key).value)  
  
    wshell.popup 'After 2 seconds I vanish.', 2, 'Messagebox message'  
}
```

- Requires a DLL in the path and a JAR in GROOVY_HOME\lib (uses Jacob, the Java COM Bridge)

Agenda



Data are always part of the game.

- Groovy: Java on dope
- Builders
- Groovy on Windows
- What about performance?
- Wrap up

Groovy performance: The price of it



```
Thread.start {1000.times {println 'Hello Thread 1'}}
Thread.start {1000.times {println 'Hello Thread 2'}}
Thread.start {1000.times {println 'Hello Thread 3'}}
Thread.start {1000.times {println 'Hello Thread 4'}}
```

- Java 71-82ms
- Groovy 220-240ms, 3x slower
- → The dynamic and reflective nature makes Groovy considerably slower than Java

```
public class ThreadTest
{
    public static void main(String[] args)
    {
        new Thread() {
            public void run()
            {
                for (int i = 0; i < 1000; i++)
                    System.out.println("Hello Thread 1");
            }
        }.start();

        new Thread() {
            public void run()
            {
                for (int i = 0; i < 1000; i++)
                    System.out.println("Hello Thread 2");
            }
        }.start();

        new Thread() {
            public void run()
            {
                for (int i = 0; i < 1000; i++)
                    System.out.println("Hello Thread 3");
            }
        }.start();

        new Thread() {
            public void run()
            {
                for (int i = 0; i < 1000; i++)
                    System.out.println("Hello Thread 4");
            }
        }.start();
    }
}
```

Groovy performance: how come?



```
println this.class.methods.name
```

...compiles to:

```
public Object run() {  
    Class class1 = GPathSimple.class;  
    Class class2 = groovy.lang.MetaClass.class;  
    return ScriptBytecodeAdapter  
        .invokeMethodOnCurrentN  
        (class1,  
         (GroovyObject)ScriptBytecodeAdapter.castToType(this, groovy.lang.GroovyObject.class),  
         "println",  
         new Object[] {  
             ScriptBytecodeAdapter.getProperty  
                 (class1,  
                  ScriptBytecodeAdapter.getProperty  
                      (class1,  
                       ScriptBytecodeAdapter.getGroovyObjectProperty(class1, this, "class"),  
                       "methods"),  
                       "name")  
         }  
        });  
}
```

Groovy performance: Good news



- Groovy 1.6 promises 100x speed-up in regard of dynamic calls

- JSR 292 adds a new 'invokedynamic' instruction to the JVM
 - Scheduled for Java 7

Agenda



Data are always part of the game.

- Groovy: Java on dope
- Builders
- Groovy on Windows
- What about performance?
- Wrap up

Groovy drawbacks (are there any...?)



- 2..10.times { slower } // than Java
- The dynamic nature lacks of compile time security
 - No safe refactorings
 - Less useful code completion in IDEs
 - Scripts: No need to declare variables lead to hard-to-find errors
 - → Test, test, test
- The resolver behaviour is sometimes tricky to understand
 - e.g.: `this.class.methods.name` // name is a prop of a single method
 - e.g.: `println map[key]` // is `,key`` a literal string or a variable?
 - No semicolons → Problem with multiline expressions
- The last expression is automatically returned
 - → Maybe not what I really want

Groovy advantages



- Very expressive
- Short syntax
- Easy SQL
- Easy trees with builders
- Easy integration (e.g. with windows)
- Extends Java classes with functionality we always wished to have
- Sometimes more compile time security when writing config files with builders instead of XML

Things that didn't made it into this presentation



```
sql = new Sql(datasource)
sql.eachRow('select * from product_t') {
    println "We sell the ${it.name_long} at \${it.price}"
}
```

■ Groovy XML

■ Various goodies

- Spread operator `*`
- Elvis operator `?:` (this is *not* `<cond> ? <result1> : <result2>`)
- Method reference operator `.&`
- Spaceship operator `<=>`
- `File.withWriter { ... }`
- RegEx-handling and –improvements
- Reverse ranges, exclusive ranges (`..>`)
- `switch` statement and the `isCase()` method

Things that didn't made it into this presentation II



- GORM: Groovy Object Relational Mapping
 - Is developed under the hoods of Grails
- Dynamically creating and executing classes and scripts
- Working with the MetaClass
- Groovlets (Groovy Servlets)
- Groovy Templating
- Groovy WebServices
 - SOAP-Support
 - REST-Support
 - XML-RPC

Things that didn't made it into this presentation III



- Groovy and Ant:
 - AntBuilder,
 - Groovy-task for Ant,
 - excuting AntScripts *within* Groovy

- Groovy and Spring
 - BeanBuilder
 - `groovy` namespace in Spring XML

- Groovy and Java
 - Just works as expected
 - ...except some class loading issues with various server configurations

- Debugging
 - Currently only if you have plenty of time

■ ■ ■ Thank you!



?

www.trivadis.com

trivadis
makes IT easier. ■ ■ ■