Dynamic meta-programming and Domain-specific languages

Václav Pech

NPRG014 2025/2026

http://www.vaclavpech.eu

@vaclav_pech

Last time agenda

Scripting

Functional programming

- Groovy syntax and interoperability
- Language dynamism

Agenda for today

- Dynamic meta-programming
- Builders
- Domain specific languages

Today's agenda

You will learn:

How the dynamic dispatch mechanism works

How to utilize it to alter the run-time behavior of a class

What are builders and how to build them

How to create text-like DSLs

Review

Groovy syntax and interoperability

Power assert

```
assert 5 == customer.score
```

```
Exception thrown
17.2.2012 12:30:12 org.codehaus.groovy.runtime.StackTraceUtils sanitize

WARNING: Sanitizing stacktrace:

Assertion failed:

assert 5 == customer.score
         |  |        |
         |  |        4
         |  [score:4]
         false
```

Groovy is functional

```
def multiply = {a, b -> a * b}
def double = multiply.curry(2)
def triple = multiply.curry(3)

assert 4 == multiply(2, 2)
assert 8 == double(4)
assert 6 == triple(2)
```

Closure scope

owner

delegate

this

Collections

```
final emptyList = []
final list = [1, 2, 3, 4, 5]
final emptyMap = [:]
final capitals = [cz : 'Prague', uk : 'London']

final list = [1, 2, 3, 4, 5] as LinkedList
final emptyMap = [:] as ConcurrentHashMap
```

Review

Scripting

Scripting

Evaluate custom Groovy code

At run-time!!!

```
new GroovyShell().evaluate('println Hi!')
```

http://groovyconsole.appspot.com/

Review

Functional programming

Functors

Dealing with wrapped data

map: ([A], f: A -> B) → [B]

map: (Maybe<A>, f: A -> B) → Maybe<B>

Functors are *mappable* (they have a **map** operation)

Monoids

Aggregating data and operations

- A set of elements
- An operation that combines two elements
- An 'id' element neutral with respect to the operation
- Closure of the set with respect to the operation

1. $a + id = id + a = a$
2. $(a + b) + c = a + (b + c)$
3. $a \in M \,\&\, b \in M \Rightarrow a+b \in M$

Monoids

Reducible – any set of elements from a monoid can be reduced into a single value

reduce: ([A], f: (A, A) → A) → A

Part 1

Dynamic meta-programming

Agenda

Dynamic dispatch

Dynamic cast

Dynamic object creation

Categories

Meta-programming

Method call dispatch

Static - the target method is decided at compile-time

Dynamic - the target method is decided at run-time

Static dispatch

At compile-time using the compile-time type of the arguments

def calculate(String value)

def calculate(Integer value)

def calculate(Object value)

Object v = 10;

calculate(v); *//Which method to call?*

Dynamic dispatch

At run-time using the run-time type of the arguments

def calculate(String value)

def calculate(Integer value)

def calculate(Object value)

Object v = 10;

calculate(v); *//Which method to call?*

Type coercion

Implicit

Runnable r = {println 'Asynchronous'}

Integer i = '42'

Explicit

def f = '3.14' as Float

def s = [1, 2, 3] as Set

Dynamic object creation

Duck-typing

```
Calculator c = [ add : {a, b → a + b},
                 multiply : {a, b → a * b},
                 increment : {it + 1}
               ] as Calculator

assert 6 == c.multiply(2, 3)
```

Traits

```
trait Flying {
    void fly() {println "I am flying!"}
}
trait Quacking {
    void quack() {println "Quack!"}
}
class Duck implements Flying, Quacking {}
```

Traits

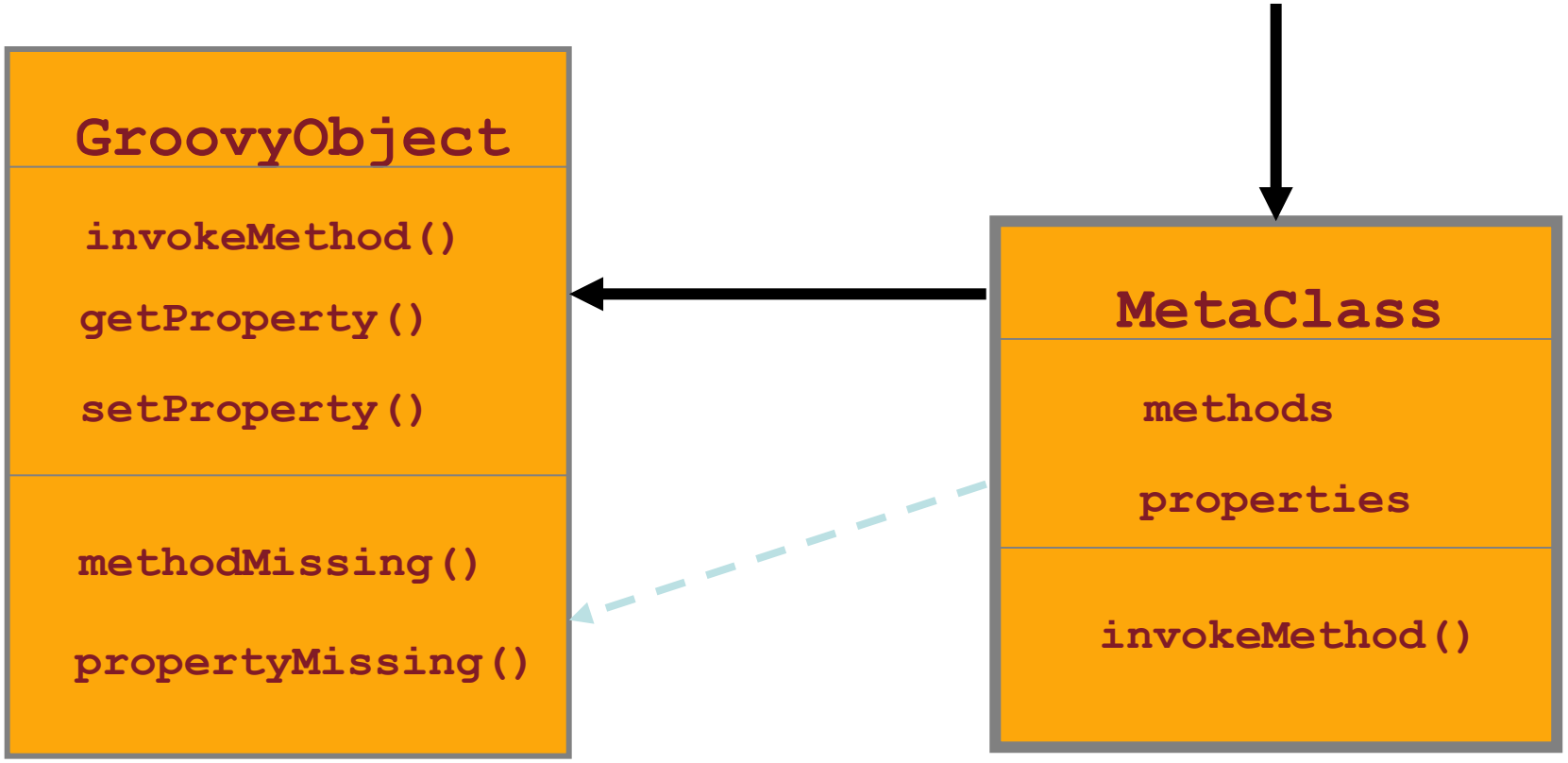
- Componentisation of the design
- Generalized delegation
- Stackable
- Can be specified as argument types

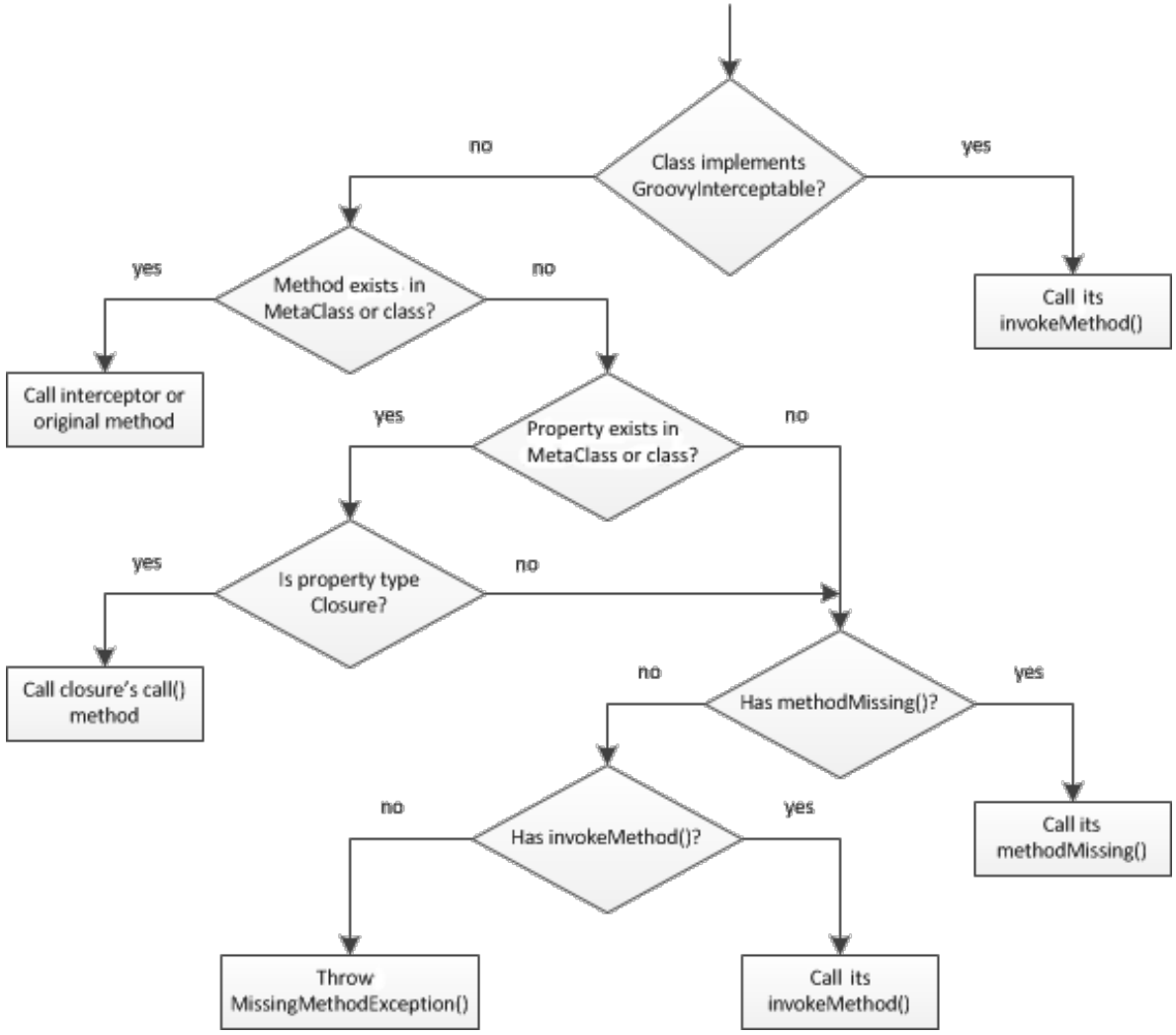
Category (extension methods)

```
StringUtils.countMatches(myString, 'Groovy')
```

```
use(StringUtils) {
    myString.countMatches('Groovy')
}
```

Dynamic method invocation




Class implements GroovyInterceptable?
no
yes
Call its invokeMethod()
Method exists in MetaClass or class?
yes
no
Call interceptor or original method
Property exists in MetaClass or class?
yes
no
Is property type Closure?
yes
no
Call closure's call() method
Has methodMissing()?
no
yes
Call its methodMissing()
Has invokeMethod()?
no
yes
Throw MissingMethodException()
Call its invokeMethod()

Querying objects' methods

o.respondsTo()

o.hasProperty()

o.metaClass.getMetaMethod(name, args)

o.metaClass.getMetaProperty(name)

Part 4

Domain Specific Languages

BDD - Spock

```
class DataDriven extends Specification {
    def "maximum of two numbers"() {
        expect:
        Math.max(a, b) == c
        where:
        a | b | c
        7 | 3 | 7
        4 | 5 | 5
        9 | 9 | 9
} }
```

https://spockframework.org/

Gradle

```
apply plugin: 'kotlin-android'

apply plugin: 'kotlin-android-extensions'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.example.myapplication"
        minSdkVersion 16
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation"org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.core:core-ktx:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.google.android.material:material:1.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

DSL

- Limited purpose language
- Targeted to a particular domain
- Friendlier API to a framework
  - External
    - SQL, HTML, CSS, …
  - Internal

DSL – Date manipulation

```
use(org.codehaus.groovy.runtime.TimeCategory) {
    println "Tomorrow: ${1.day.from.today}"
    println "A week ago: ${1.week.ago}"
    println "Date: ${1.month.ago + 1.week + 2.hours - 5.minutes}"
    println "Date ${(1.month + 10.days).ago}"
}
```

DSL – Hibernate criteria

```
def participants = Participant.createCriteria().list {
    gt('age', age)
    or{
        eq('interest', 'Java')
        eq('interest', 'Groovy')
    }
    jug {
        ilike('country', 'de')
    }
    order('lastName', 'asc')
}
```

DSL – Account manipulation

```
Money money = new Money(amount: 350, currency: 'eur')
getAccount('Account1').withDraw money
getAccount('Account3').deposit money
```

```
"Account1" >> 350.eur >> "Account3"
```

Why DSLs

Domain expressiveness – code closely matches domain concepts, improving readability.

Reduced boilerplate – conciseness compared to general-purpose language constructs.

Improved communication – shared language between developers and domain experts.

Higher abstraction – hides low-level details, focusing on "what" not "how."

Consistency & correctness – enforces domain-specific rules directly in syntax.

Rapid prototyping – easier experimentation with domain models and solutions.

Integration leverage – benefits from host language tooling (IDE, debuggers, libraries).

Easier maintenance – domain logic is explicit, reducing cognitive load over time.

Example – file access DSL

```
File.metaClass.div = { path ->
    new File(delegate, path)
}
```

Example – file access DSL

```
File.metaClass.div = { path ->
    new File(delegate, path)
}
def file = new File(".")/'test'/'hello'/'file.txt'
```

order cake with plums and apples
and cream to "Malostranske namesti"

```
order(cake).with(plums).and(apples)
.and(cream).to("Malostranske namesti")
```

Builders

Construct hierarchies

- html, xml, json, swing, configuration
- objects
- db queries
- ...

Builders - GAnt

```
ant.sequential {
    myDir = "target/AntTest/"
    mkdir(dir: myDir)
    copy(todir: myDir) {
        fileset(dir: "src/test") {
            include(name: "**/*.groovy")
        }
    }
    List dirs = ['core', 'lib', 'engine', 'gui', 'db']
    for(String currentDir:dirs) {
        String targetDir="target/$currentDir"
        mkdir(dir:targetDir)
```

Cli Builder

```groovy
def cli = new CliBuilder (usage:'simpleHtmlServer -p PORT -d DIRECTORY')
cli.with {
 h longOpt:'help', 'Usage information'
 p longOpt:'port',argName:'port', args:1, type:Number.class,'Default is 8080'
 d longOpt:'dir', argName:'directory', args:1, 'Default is .'
}

def opts = cli.parse(args)
if(!opts) return
if(opts.help) {
  cli.usage()
  return
}
```

Builders – Spring config

```
dataSource(BasicDataSource) {
    driverClassName = "org.hsqldb.jdbcDriver"
    url = "jdbc:hsqldb:mem:shopDB"
}

sessionFactory(ConfigurableLocalSessionFactoryBean) {
    dataSource = dataSource
    hibernateProperties = ["hibernate.hbm2ddl.auto": "create-drop",
            "hibernate.show_sql": true]
}

calculator(demo.shop.CalculatorImpl) {bean ->
    bean.singleton = true
    bean.autowire = 'byType'
}
```

Summary

Dynamic method call dispatch
Dynamic object creation
Builders
Traits
Categories
Dynamic DSLs

References

http://groovy-lang.org

http://grails.org

https://martinfowler.com/books/dsl.html