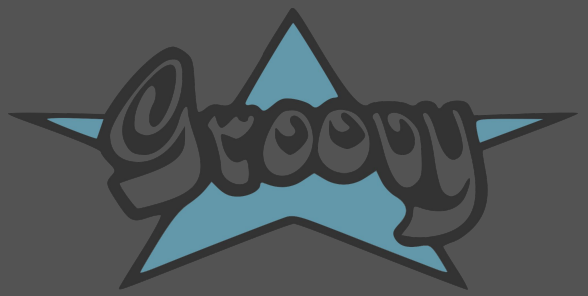
Groovy

Apache Groovy with Gradle

Óbudai Egyetem, Java Enterprise Edition
Műszaki Informatika szak
Labor 10

Bedők Dávid
2016.10.13.
v0.7

Groovy



http://www.groovy-lang.org/

Latest: **2.4.5**

JVM Required: 1.7+

Apache Groovy (from 2.4.4)

- A multi-faceted language for the Java platform
- Optionally typed and dynamic language
- Static-typing and static compilation capabilities

Install: unzip

(System) Environment Variable:

GROOVY_HOME - c:\qapps\groovy-2.4.5\

Path kiegészítése: %GROOVY_HOME%\bin

```
>groovy --version
Groovy Version: 2.4.5
JVM: 1.8.0_20 Vendor:
Oracle Corporation OS:
Windows 7
```

Hello World

hello.groovy

```
def name = 'Qwaevisz'
def greeting = "Hello ${name}"
println greeting
```

GString (double-quoted string literals, **interpolation**): Groovy will auto-cast between GString and String

```
>groovy hello
```

```
Hello Qwaevisz
```

Ismerkedés Java mellett

Ki lehet indulni abból a feltevésből, hogy ha egy adott problémát Java nyelven le tudunk kódolni (oo elvek mentén), akkor durván azonos szintaktikával mindez groovy script formájában is működni fog.

Egy Eclipse Groovy plugin segítségével folyamatosan tudjuk az így megírt Groovy osztályokat “groovy-sítani”, ezáltal megtanulni a nyelv valós használatát.

- **Diamond operator** (<>) létezik kompatibilitási okokból, de teljesen haszontalan a dinamikus Groovy-ban (pl.: `List<String> strings = new LinkedList<>()`)(the Groovy type checker performs type inference whether this operator is present or not).
- **Optional parentheses**: Method calls can omit the parentheses if there is at least one parameter and there is no ambiguity.
- **Optional semicolons**: In Groovy semicolons at the end of the line can be omitted, if the line contains only a single statement.
- **Optional return keyword**: In Groovy, the last expression evaluated in the body of a method or a closure is returned. This means that the return keyword is optional.
- **Optional public keyword**: By default, Groovy classes and methods are public.

Script vs. Application

Main.groovy

```
class Main {
    static void main(String... args) {
        println 'Groovy world!'
    }
}
```

script.groovy

```
println 'Groovy world!'
```

A Script automatikus átforog az itt látható osztályba.

```
import org.codehaus.groovy.runtime.InvokerHelper
class Main extends Script {
    def run() {
        println 'Groovy world!'
    }
    static void main(String[] args) {
        InvokerHelper.runScript(Main, args)
    }
}
```

Runtime dispath

In **Groovy**, the methods which will be invoked are chosen at **runtime**. This is called **runtime dispatch** or **multi-methods**. It means that the method will be chosen based on the types of the arguments at runtime. In **Java**, this is the opposite: methods are chosen at **compile time**, based on the declared types.

```
int method(String arg) {
    return 1;
}
int method(Object arg) {
    return 2;
}
Object o = "Object";
int result = method(o);
```

JAVA

```
assertEquals(2, result);
```

GROOVY

```
assertEquals(1, result);
```

Eclipse plugin

Update site: http://dist.springsource.org/snapshot/GRECLIPSE/e4.5/

- **src/main/groovy** alá helyezzük el a groovy scriptjeinket
- Nem kell a package név-osztály név szabályt betartani, illetve egy file-ban több nyilvános osztály is helyet kaphat. Azonban ezt kerüljük el mixed (java+groovy) projektek esetén!

Alapértelmezett import-ok

Minden Groovy scriptre érvényes:

- java.io.*
- java.lang.* *(Java esetén csak ez érvényes)*
- java.math.BigDecimal
- java.math.BigInteger
- java.net.*
- java.util.*
- groovy.lang.*
- groovy.util.*

Új kulcsszavak

- **as**
  - we can **change the type** of objects with the as keyword in Groovy
  - we can even make maps and closure look like interface implementations when we use as
  - furthermore we can use as to **create an alias for an import** statement
- **def**
  - def is a **replacement for a type name**
  - in variable definitions it is used to indicate that you **don’t care about the type**
  - you can think of def as an **alias of Object**
  - omitting the "def" keyword: like a globally scoped variable
- **in**
  - used in Groovy to **check whether element exists in Collection**
  - membership operator
- **trait**
  - **a structural construct** of the language
  - can be seen as **interfaces carrying both default implementations and state**

Literálok

Integral literals (numbers)

- byte
- char
- short
- int
- long
- java.lang.BigInteger

Decimal literals

- float
- double
- java.lang.BigDecimal

Booleans

```
def myBooleanVariable = true
boolean untypedBooleanVar = false
booleanField = true
```

Új operátorok I.

Elvis operator (?:)

```
displayName = user.name ? user.name : 'Anonymous'
displayName = user.name ?: 'Anonymous'
```

Safe navigation operator (?.)

```
def person = null
def name = person?.name
assert name == null
```

use of the null-safe operator prevents from a NullPointerException

Direct field access operator (.@)

```
class User {
    public final String name
    User(String name) { this.name = name}
    String getName() { "Name: $name" }
}
def user = new User('Bob')
assert user.name == 'Name: Bob'
assert user.@name == 'Bob'
```

The user.name call triggers a call to the property of the same name, that is to say, here, to the getter for name.

Új operátorok II.

Pattern operator (~)

- The pattern operator (~) provides a simple way to create a java.util.regex.Pattern instance.

Find operator (=~)

- Alternatively to building a pattern, you can directly use the find operator =~ to build a java.util.regex.Matcher instance.

Match operator (==~)

- The match operator (==~) is a slight variation of the find operator, that does not return a Matcher but a boolean and requires a strict match of the input string.

Membership operator (in)

```
def list = ['Grace','Rob','Emmy']
assert ('Emmy' in list)
```

equivalent to calling list.**contains**('Emmy') or list.**isCase**('Emmy')

Új operátorok - Method pointer (.&)

Method pointer operator (.&)

```
def str = 'example of method reference'
def fun = str.&toUpperCase
def upper = fun()
assert upper == str.toUpperCase()
```

a method pointer is a groovy.lang.**Closure**

mpointer.groovy

```
def transform(List elements, Closure action) {
    def result = []
    elements.each {
        result << action(it)
    }
    result
}
String describe(Person p) {
    "$p.name is $p.age"
}
def action = this.&describe
def list = [
    new Person(name: 'Bob',   age: 42),
    new Person(name: 'Julia', age: 35)]
assert transform(list, action) == ['Bob is 42', 'Julia is 35']
```

Új operátorok - Spaceship (<=>)

Method pointer operator (.&)

mpointer.groovy

```
def doSomething(String str) { str.toUpperCase() }
def doSomething(Integer x) { 2*x }
def reference = this.&doSomething
assert reference('foo') == 'FOO'
assert reference(123)   == 246
```

Method pointers are bound by the receiver and a method name. Arguments are resolved at runtime!

Spaceship operator (<=>)

spaceship.groovy

```
assert (1 <=> 1) == 0
assert (1 <=> 2) == -1
assert (2 <=> 1) == 1
assert ('a' <=> 'z') == -1
```

The spaceship operator (<=>) delegates to the **compareTo** method.

Új operátorok - Spread (*., *, *:)

The Spread Operator (*.) is used to invoke an action on all items of an aggregate object. It is equivalent to calling the action on each item and collecting the result into a list.

spread.groovy

```
class Car {
    String make
    String model
}

def cars = [
        new Car(make: 'Peugeot', model: '508'),
        new Car(make: 'Renault', model: 'Clio')]
def makes = cars*.make
assert makes == ['Peugeot', 'Renault']
```

The spread operator is null-safe, meaning that if an element of the collection is null, it will return null instead of throwing a NullPointerException

```
def items = [4,5]
def list = [1,2,3,*items,6]
assert list == [1,2,3,4,5,6]
```

```
def m1 = [c:3, d:4]
def map = [a:1, b:2, *:m1]
assert map == [a:1, b:2, c:3, d:4]
```

Új operátorok - Range (.. és ..<)

range.groovy

```
def range = 0..5
assert (0..5).collect() == [0, 1, 2, 3, 4, 5]
assert (0..<5).collect() == [0, 1, 2, 3, 4]
assert (0..5) instanceof List
assert (0..5).size() == 6
assert ('a'..'d').collect() == ['a','b','c','d']
```

Új operátorok - is/as

Identity operator (==)

In Groovy, using == to test equality is different from using the same operator in Java. In Groovy, it is calling equals.

Compare reference equality operator (is)

is.groovy

```
def list1 = ['Groovy 1.8','Groovy 2.0','Groovy 2.3']
def list2 = ['Groovy 1.8','Groovy 2.0','Groovy 2.3']
assert list1 == list2
assert !list1.is(list2)
```

Coercion operator (as)

as.groovy

```
Integer x = 123
String s = x as String
```

Integer is not assignable to a String, so `String s = (String) x` will produce a ClassCastException at runtime.

Advanced Coercion operator

advancedas.groovy

```
interface Greeter {
    void greet(String name)
}

class DefaultGreeter {
    void greet(String name) { println "Hello" }
}

greeter = new DefaultGreeter()
assert !(greeter instanceof Greeter)
coerced = greeter as Greeter
assert coerced instanceof Greeter
```

An object can **implement an interface at runtime**, using the `as` coercion operator.

Operator overloading

opoverload.groovy

```
class Bucket {
    int size

    Bucket(int size) { this.size = size }

    Bucket plus(Bucket other) {
        return new Bucket(this.size + other.size)
    }
}

def b1 = new Bucket(4)
def b2 = new Bucket(11)
assert (b1 + b2).size == 15
```

Minden operátornak létezik a “logikus” metódus párja.

Vezérlési szerkezetek - switch case

switch.groovy

```groovy
def x = 1.23
def result = ""
switch ( x ) {
    case "foo":
        result = "found foo"
    case "bar":
        result += "bar"
    case [4, 5, 6, 'inList']:
        result = "list"
        break
    case 12..30:
        result = "range"
        break
    case Integer:
        result = "integer"
        break
    case Number:
        result = "number"
        break
```

switch.groovy

```groovy
    case ~/fo*/:
        result = "foo regex"
        break
    case { it < 0 }: // { x < 0 }
        result = "negative"
        break

    default:
        result = "default"
}
```

Vezérlési szerkezetek - Ciklusok

for.groovy

```
def x = 0
for ( i in 0..9 ) {
    x += i
}
assert x == 45

// iterate over a map
def map = ['abc':1, 'def':2,
'xyz':3]
x = 0
for ( e in map ) {
    x += e.value
}
assert x == 6
```

for.groovy

```
// iterate over values in a map
x = 0
for ( v in map.values() ) {
    x += v
}
assert x == 6

// iterate over the characters in a
string
def text = "abc"
def list = []
for (c in text) {
    list.add(c)
}
assert list == ["a", "b", "c"]
```

Groovy also supports the **Java colon variation** with colons: `for (char c : text) {}`, where the type of the variable is mandatory.

Különbségek I.

JAVA | GROOVY

- Array initializers ({ … } block is reserved for closures)

```
int[] array = {1,2,3}
```

```
int[] array = [1,2,3]
```

- Ommit modifiers

```
package private (default)
```

```
public / property
(use @PackageScope for Java
default)
```

Groovy **property**: a private field, an associated getter and an associated setter

- Behaviour of ==

In Java == means equality of primitive types or identity for objects.

In Groovy == translates to a.compareTo(b)==0, if they are Comparable, and a.equals(b) otherwise. To check for identity, there is is. E.g. a.is(b).

Primitives

Because Groovy uses Objects for everything, it autowraps references to primitives. Because of this, it does not follow Java's behavior of widening taking priority over boxing.

```
int i
m(i)

// Java use that
void m(long l) {
  println "in m(long)"
}

// Groovy use that
void m(Integer i) {
  println "in m(Integer)"
}
```

Listák, tömbök

- Groovy lists are plain JDK **java.util.List**, as Groovy doesn't define its own collection classes
- actually instances of **java.util.ArrayList**

list.groovy

```
def numbers = [1, 2, 3]
def heterogeneous = [1, "a", true]
def letters = ['a', 'b', 'c', 'd']
letters[2] = 'C'
letters << 'e'

def multi = [[0, 1], [2, 3]]
println multi[1][0]
assert [1, 2, 3, 4, 5][-2] == 4
```

use negative indices to count from the end

array.groovy

```
String[] arrStr = ['Ananas', 'Banana', 'Kiwi']
def numArr = [1, 2, 3] as int[]
```

Map

map.groovy

```
def colors = [red: '#FF0000', green: '#00FF00', blue:
'#0000FF']
println colors['red']
println colors.green
colors.containsKey('yellow')
```

When you need to pass variable values as keys in your map definitions, you must surround the variable or expression with parentheses

map.groovy

```
def key = 'red'
def colors = [(key): '#FF0000']
```

Triple single quoted strings és Import aliasing

Triple single quoted strings are plain java.lang.String and don’t support interpolation.

Triple single quoted strings are multiline.

```
'''a triple single quoted string'''
```

Static import aliasing

```
import static Calendar.getInstance as now
assert now().class == Calendar.getInstance().class
```

Import aliasing

```
import thirdpartylib.MultiplyTwo as OrigMultiplyTwo
```

Groovy truth

- non empty string, evaluating to true

```
assert(!'foo') == false
```

- " is an empty string, evaluating to false, so negation returns true

```
assert(!'') == true
```

- non-empty Collections are true
- true if the Matcher has at least one match
- non-empty Maps are evaluated to true
- non-empty Strings, GStrings and CharSequences are coerced to true
- non-zero numbers are true
- non-null object references are coerced to true

Egyedi “Groovy igazság”

truth.groovy

```
class Color {
    String name

    boolean asBoolean(){
        name == 'green' ? true : false
    }
}

assert new Color(name: 'green')
assert !new Color(name: 'red')
```

ternary.groovy

```
result = (string!=null && string.length()>0) ? 'Found' :
'Not found'
result = string ? 'Found' : 'Not found'
```

Objektum-orientáltság (különbségek)

- Public fields are turned into properties automatically
- an interface only defines method signatures
- void helyett def
  - Methods in Groovy always return some value. If no return statement is provided, the value evaluated in the last line executed will be returned.
- It is possible to omit the type declaration of a field (but this is bad practice)
- Method Default arguments
  - Note that no mandatory parameter can be defined after a default par
    parameters.

```
def foo(String par1, Integer par2 = 1) {..}
```

Constructor

constructor.groovy

```
class Person {
    String name
    Integer age

    Person(name, age) {
        this.name = name
        this.age = age
    }
}

def person1 = new Person('Marie', 1)
def person2 = ['Marie', 2] as Person
Person person3 = ['Marie', 3]
```

Named Argument Constructor

naconstructor.groovy

```
class Person {
    String name
    Integer age
}

def person1 = new Person()
def person2 = new Person(name: 'Marie')
def person3 = new Person(age: 1)
def person4 = new Person(name: 'Marie', age: 2)

println person4.properties
println person4.properties.keySet()
```

```
{class=class hu.qwaevisz.hellogroovy.Person, age=2, name=Marie}
```

```
[class, age, name]
```

Trait - Jellemvonás

They can be seen as interfaces carrying both default implementations and state.

trait.groovy

```
trait FlyingAbility {
        String fly() { "I'm flying!" }
}

class Bird implements FlyingAbility {}
def b = new Bird()
assert b.fly() == "I'm flying!"
```

When coercing an object to a trait, the result of the operation is not the same instance. It is guaranteed that the coerced object will implement both the trait and the interfaces that the original object implements, but the result will not be an instance of the original class.

Trait

- In addition, traits may declare **abstract methods too**, which therefore need to be implemented in the class implementing the trait!
- Traits can be used to implement **multiple inheritance** in a controlled way, avoiding the diamond issue.
- Traits may also define **private methods**.
- Traits may **implement interfaces**, in which case the interfaces are declared using the **implements** keyword.
- Traits may **extend another trait**, in which case you must use the **extends** keyword.
  - BUT: Multiple inheritance: Alternatively, a trait may extend multiple traits. In that case, all super traits must be declared in the implements clause (!!!).
- A trait may define **properties**.
- Since traits allow the use of private methods, it can also be interesting to use **private fields** to store state.
- Public fields work the same way as private fields, but in order to avoid the diamond problem, field names are remapped in the implementing class (BAD PRACTICE) (Átnevezés során hozzácsapja a csomag és a mező nevét, pl.: my_package_Foo__bar)

Closure

- A closure in Groovy is an open, anonymous, block of code that can take arguments, return a value and be assigned to a variable.
- A closure is an instance of the `groovy.lang.Closure` class.
- Unlike a method, a closure always returns a value when called.

closure.groovy

```
{ String x, int y -> println "hey ${x} the value is ${y}" }
def code = { 123 }
assert code() == 123
assert code.call() == 123

def isOdd = { int i -> i%2 == 1 }
assert isOdd(3) == true
assert isOdd.call(2) == false

def greeting = { "Hello, $it!" }
assert greeting('Patrick') == 'Hello, Patrick!'
```

Implicit parameter: When a closure does not explicitly define a parameter list (using ->), a closure always defines an implicit parameter, named `it`.

Closures are at the core of the functional programming paradigm in Groovy.

Továbbiak I.

Expression Deconstructions

ed.groovy

```
this.class  --> this.getClass()
this.class.methods --> this.getClass().getMethods()
this.class.methods.name --> ...
this.class.methods.name.grep(… )
```

String to Enum coercion

s2e.groovy

```
enum State {
    up,
    down
}
State st = 'up'
assert st == State.up
```

Továbbiak II.

Call operator

call.groovy

```
class MyCallable {
    int call(int x) {
        2*x
    }
}
def mc = new
MyCallable()
assert mc.call(2) == 4
assert mc(2) == 4
```

The call operator () is used to call a method named call implicitly.

PseudoProperties

pseudo.groovy

```
class PseudoProperties {
  void setName(String name) {}
  String getName() {}

  int getAge() { 42 }

  void setGroovy(boolean groovy) {}
}

def p = new PseudoProperties()
p.name = 'Foo'
assert p.age == 42
p.groovy = true
```

Meta-Annotation

CustomTransationalService.groovy

```
@Service
@Transactional
class CustomTransactionalService {}
```

CustomTransationalService.groovy

```
@TransactionalService
class CustomTransactionalService {}
```

TransactionalService.groovy

```
import groovy.transform.AnnotationCollector

@Service
@Transactional
@AnnotationCollector
@interface TransactionalService {
}
```

Dynamic methods (MOP methods)
Meta-Object-Protocol

dm.groovy

```
String methodMissing(String name, args) {
    "${name.capitalize()}!"
}
def propertyMissing(String prop) {
    props[prop]
}
```

Monkey patching

meta.groovy

```
class Person {
    String firstName
    String lastName
}
def p = new Person(firstName: 'Raymond', lastName: 'Devos')
Person.metaClass.getFormattedName = { "$delegate.firstName
$delegate.lastName" }
assert p.formattedName == 'Raymond Devos'
```

Recursive closure és memoize()

fibonacci.groovy

```
def fib
fib = { long n -> n<2 ? n : fib(n-1) + fib(n-2) }
assert fib(15) == 610 // slow!


fib = { long n -> n<2 ? n : fib(n-1) + fib(n-2) }.memoize()
assert fib(25) == 75025 // fast!
```

A memoize() a már kiszámolt értékeket “cache”-ben tárolja, ezzel megspórolva azok ismételt kiszámolását.

Advanced List

each, collect, find, indexOf, sum, join, max, min

list.groovy

```
[1, 2, 3].each { println "Item: $it" }
['a', 'b', 'c'].eachWithIndex { it, i -> println "$i: $it" }
assert [1, 2, 3].collect { it * 2 } == [2, 4, 6]
assert [1, 2, 3].find { it > 1 } == 2
assert [1, 2, 3].findAll { it > 1 } == [2, 3]
assert ['a', 'b', 'c', 'd', 'e'].findIndexOf { it in ['c', 'e', 'g'] } == 2
assert ['a', 'b', 'c', 'd', 'c'].indexOf('c') == 2
assert ['a', 'b', 'c', 'd', 'c'].indexOf('z') == -1
assert ['a', 'b', 'c', 'd', 'c'].lastIndexOf('c') == 4
assert [1, 2, 3].every { it < 5 } // returns true if all elements match
assert [1, 2, 3].any { it > 2 } // returns true if any element matches
assert [1, 2, 3, 4, 5, 6].sum() == 21
assert [1, 2, 3].sum(1000) == 1006 // initial value
assert [1, 2, 3].join('-') == '1-2-3'
def list = [9, 4, 2, 10, 5]
assert list.max() == 10
assert list.min() == 2

// we can use a closure to specify the sorting behaviour
def list2 = ['abc', 'z', 'xyzuvw', 'Hello', '321']
assert list2.max { it.size() } == 'xyzuvw'
assert list2.min { it.size() } == 'z'
```

Advanced Map

each, collect, find, indexOf, sum, join, max, min

map.groovy

```
def people = [
    1: [name:'Bob', age: 32, gender: 'M'],
    2: [name:'Johnny', age: 36, gender: 'M'],
    3: [name:'Claire', age: 21, gender: 'F'],
    4: [name:'Amy', age: 54, gender:'F']
]

def bob = people.find { it.value.name == 'Bob' }
def females = people.findAll { it.value.gender == 'F' }

def agesOfMales = people.findAll { id, person ->
    person.gender == 'M'
}.collect { id, person ->
    person.age
}
assert agesOfMales == [32, 36]
```

Type checked mode

`@TypeChecked`

- Osztályon/metóduson elhelyezhető
- When type checking is activated, the compiler performs much more work.

`@TypeChecked(TypeCheckingMode.SKIP)`

- Ha az osztályon van @TypeChecked, akkor metóduson ezzel kikapcsolható.

`@CompileStatic`

- Type safety
- Immunity to monkey patching
- Performance improvements

Hello Groovy

build.gradle

```
apply plugin: 'groovy'
apply plugin: 'eclipse'

sourceCompatibility = 1.7
version = '1.0'

sourceSets.main.java.srcDirs = []
sourceSets.main.groovy.srcDirs = ['src/main/java',
'src/main/groovy']

repositories { mavenCentral() }

def groovyVersion = '2.4.5'

dependencies {
   compile group: 'org.codehaus.groovy', name: 'groovy' ,
version: groovyVersion
}
```

```
>gradle clean build
```

A java kódot önmagában ne fordítsa le, a groovy-t viszont a Java-val együtt (mixed mód)

Artifactory

Alap

- `org.codehaus.groovy:groovy:x.y.z`

Module

- `org.codehaus.groovy:groovy-$module:x.y.z`
- `$module`: ant, bsf, console, docgenerator, groovydoc, groovysh, jmx, json, jsr223, nio, servlet, sql, swing, test, templates, testng, xml

Összes

- `org.codehaus.groovy:groovy-all:x.y.z`

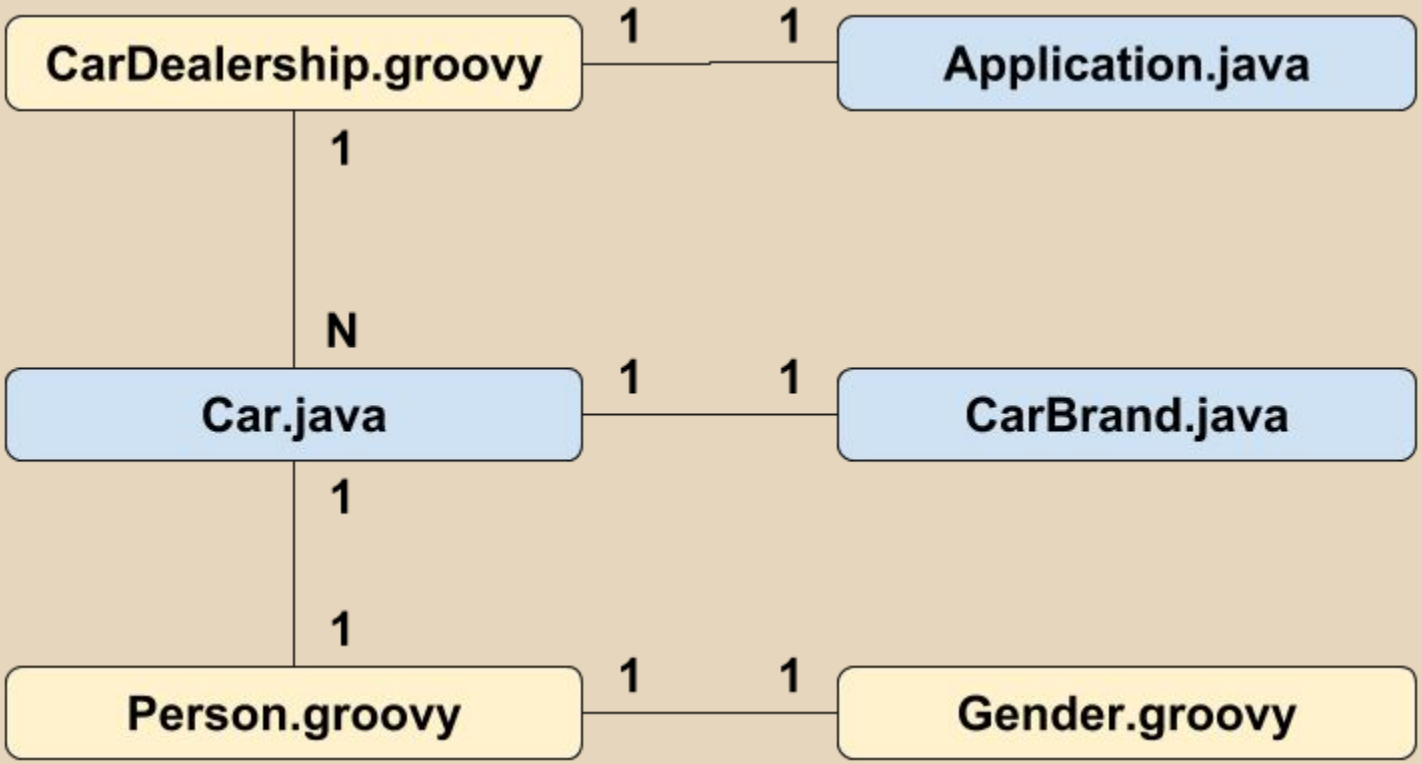
Mixed project

Teljesen Groovy projektet is lehet készíteni, azonban feltehetően nagyobb jövője van a történetnek ha a Groovy-t egy meglévő Java projekt kiegészítéseként használjuk.

A projekt bizonyos részeit Java-ban, bizonyosakat Groovy-ban készítünk el. Mindegyikből byte-code készül, futás közben a “forrásnyelv” már lényegtelenné válik.

Mind a Java kód hívhat Groovy kódot, és a Groovy kód is hívhat Java kódot.

Osztályok


CarDealership.groovy
1
1
Application.java
1
N
Car.java
1
1
CarBrand.java
1
1
Person.groovy
1
1
Gender.groovy

Személy

Person.groovy

```
class Person {

    def familyName
    def firstName
    String identifier
    Gender gender

    ..

    String toString() {
        '(' + [
            "familyName" : this.familyName,
            "firstName" : this.firstName,
            "identifier" : this.identifier,
            "gender" : this.gender
        ].iterator().join(', ') + ')'
    }
}
```

Kulcs-érték párok listája, mely össze van fűzve vesszőkkel. Eredmény:

```
(familyName=Anakin,
firstName=Skywalker,
identifier=FJFEMV56,
gender=MAN)
```

Autó kereskedés

CarDealership.groovy

```
class CarDealership {
    private List cars

    CarDealership() { this.cars = [] }

    def add( Car car ) { this.cars << car }

    Car find( CarBrand brand, String model ) {
        this.cars.find({ it -> it.brand == brand &&
it.model.startsWith(model)})
    }

    @Override
    String toString() {
        cars
    }
}
```

Java-ban mindez pár karakterrrel több lenne...

Gradle és Groovy

version.gradle

```
import groovy.json.JsonSlurper
class HelloVersion {
    int major
    int minor
    def build
    @Override
    String toString() {
        major + '.' + minor + build
    }
}
class HelloVersionLoader {
    def HelloVersion version
    HelloVersionLoader() {
        def jsonSlurper = new JsonSlurper()
        def content = jsonSlurper.parse(new File('version.json'))
        this.version = new HelloVersion(major: content.major, minor:
content.minor, build: content.build)
    }
}
version = new HelloVersionLoader().version
```

build.gradle

```
...
apply plugin: 'groovy'
apply from: 'version.gradle'
...
```

version.json

```
{ "major": 1, "minor": 0,
"build": "-SNAPSHOT" }
```

A `version` a gradle projekt változója.