Developing mixed Java/Groovy projects with Gradle using Netbeans

Kostas Saidis

niovity

Netbeans Day Athens 2016

August 26, 2016

About Me

Kostas Saidis
saiko@niovity.com

Software & Data Architect

Academia

- BSc @ cs.unipi.gr (2001)
- MSc @ di.uoa.gr (2004)
- PhD @ di.uoa.gr (2011)

Industry

- Freelancer since 1999 (consultant, developer & instructor)
- Enterpreneur since 2011 (niovity)

Java

- Early Java enthusiast (1997)
- Groovyist since 2011

niovity.com

We are a software company

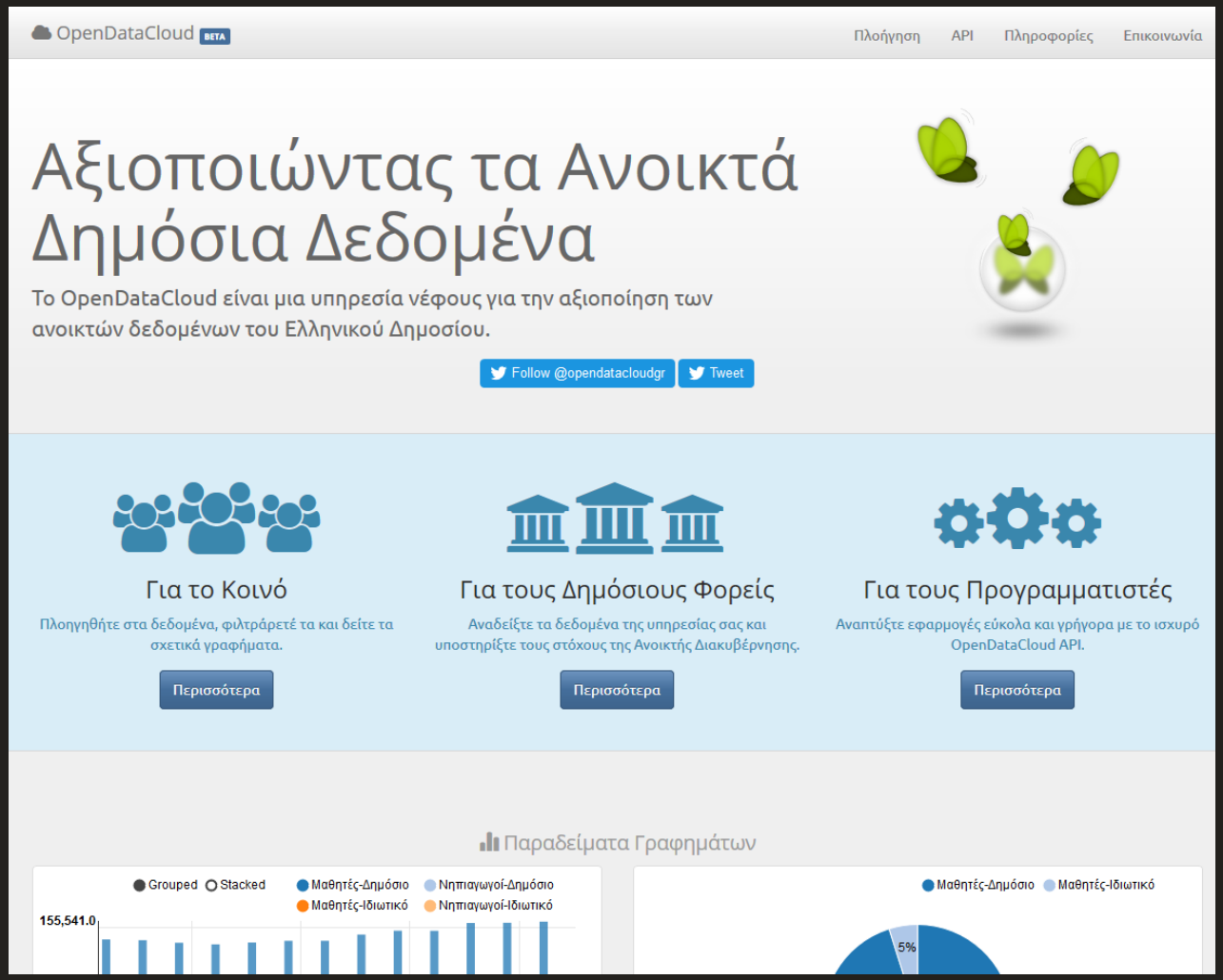
using a Java/Groovy/Javascript stack

Niovity Butterfly

We offer a web-based repository platform
for end-to-end data management

Our clients include Academic Libraries, Public Benefit Foundations, Research Institutes and Private Companies

www.OpenDataCloud.gr



2nd Prize

Public Open Data Hackathon 2014

Target audience

- [x] Java
- [x] NetBeans
- [ ] Groovy
- [ ] Gradle

2

facts

1. The Java ecosystem is polyglot now

The Java Platform

1. The Java Language
2. The Java Development Kit
3. The Java Virtual Machine (the key component)

The Java Virtual Machine

- A rock-solid, enterprise-grade runtime environment that can execute multiple languages (not only Java).

JVM Languages

- Groovy, Scala, Javascript, Ruby, Python, Clojure, Kotlin, Fantom and many others...

Groovy

The swiss army knife of the Java developer

2. Building software sucks

Software build process

- Develop
- Test
- Assemble
- Deploy
- Integrate

Repeat

- Again and again and again

Gradle

The finest build tool out there

Workshop Structure

1. Introduce Groovy & Gradle
2. Build a mixed Java/Groovy Swing application using Gradle & Netbeans

Workshop Goal

Worskhop Material and Setup Instructions

Workshop Web Page

www.niovity.com/static/NetBeansDayAthens2016/

Meet Apache Groovy

What is Groovy

A feature-rich, Java-friendly, multi-paradigm language for the JVM

- More than a decade-long history*.
- Free and Open Source Software (Apache License v2).
- Joined the Apache Software Foundation in 2015.
- #16 in the TIOBE PL Popularity Index, August 2016†.

groovy-lang.org

@groovylang #groovylang

---

*Groovy was the first "second language" proposed for the JVM (back in 2004).

†http://www.tiobe.com/tiobe-index/

A companion language for Java

- Compiles into JVM bytecode and preserves Java semantics: seamless integration with Java.
- Provides a less verbose syntax.
- Augments Java with additional features (e.g. new methods in libraries).
- Dynamic in nature, yet optional typing and compile-time static features are also supported.
- Easy to learn and use.
- Supported by all major Java IDEs[‡].

[‡] In NetBeans you should install the "All" download bundle, see here.

Why Groovy II

Community and Ecosystem

- Gradle: Build tool.
- Spock: Testing and specification framework.
- Grails: Web development framework.
- Griffon: Desktop app framework.
- And many, many more[§].

All existing Java tools, libraries and frameworks are directly reusable by Groovy.

[§]For an overview, check here.

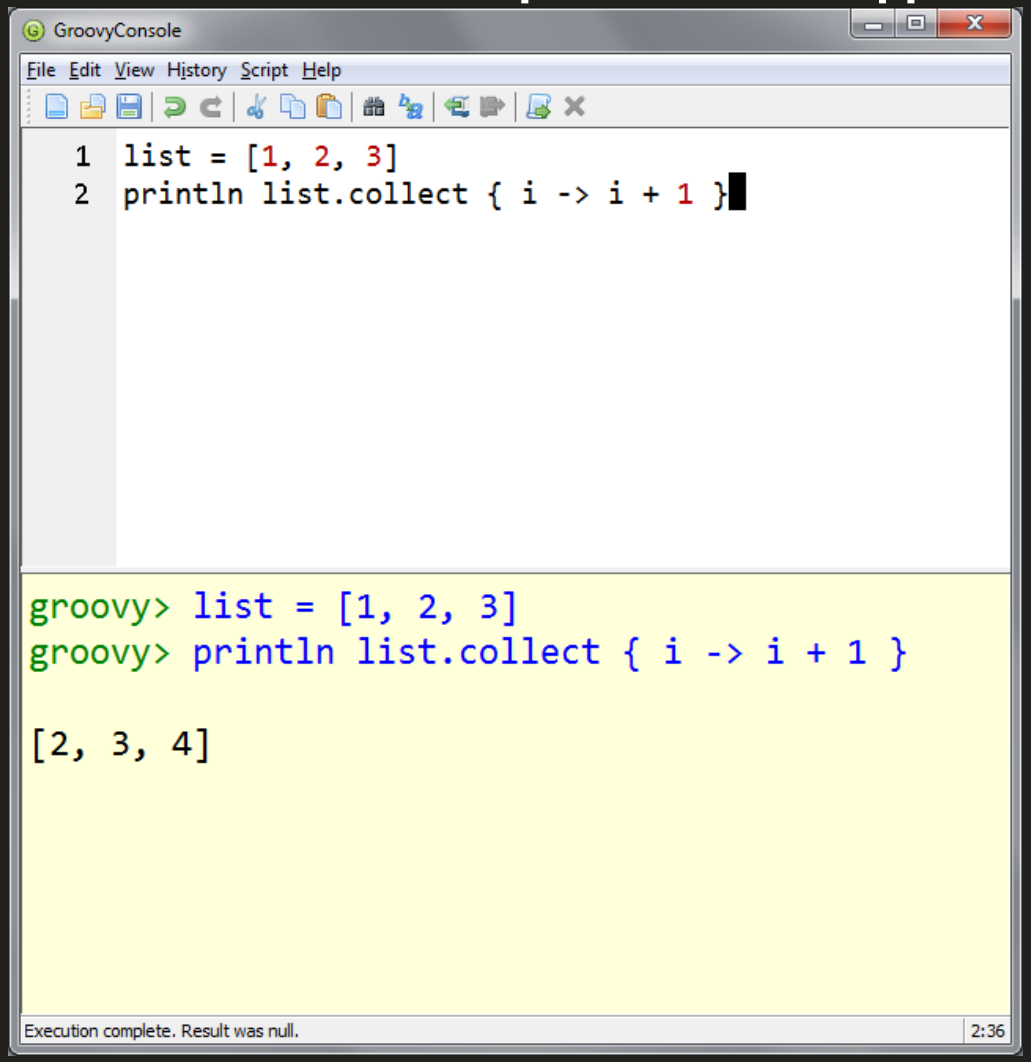
Getting Started

Groovy Tools

1. groovyc: Compiles groovy sources to JVM bytecode (class files).
2. groovysh: Executes code interactively (Read-Eval-Print Loop).
3. groovyConsole: GUI app for interactive code execution.
4. groovy: Executes groovy scripts (you can use it like bash, perl, python, etc).
5. groovydoc: Generates documentation (like javadoc).
6. grape: An embedded jar dependency manager.

The best place to start

Start from the Groovy console (GUI app)



Or the **Groovy Web Console**

The Groovy Syntax

The majority of Java syntax is part of the Groovy syntax:

- packages
- imports
- control structures
- exception handling
- classes and methods
- object instantiation and method calls

Some Gotchas: Arrays, additional Groovy keywords, equals checks, etc.

Default imports

The following imports are included by default:

- java.lang.*
- java.util.*
- java.net.*
- groovy.lang.*
- groovy.util.*
- java.math.BigInteger and java.math.BigDecimal

Optionals

The following are optional:

- semicolons: required only when you write multiple statements in one line.
- variable types: you decide what should be dynamic (usind `def`).
- return statements: the last evaluated expression is the default return value Gotcha.
- parentheses: in method/function invocations Gotcha.

Groovy truth and equals

Groovy truth

- Null value is false.
- Empty collection or map is false.
- Empty string is false.
- Zero numeric value is false.

Gotcha

- The == operator performs value equality (like Java `equals()`).
- Use the `is` method for identity (available in every Groovy object).

Examples

```
def f = null
def t = "string"
assert f == false
assert t == true
def list = []
def map = [a:1]
assert list == false
assert map == false
def s = ""
assert s == false
def i = 1
def z = 0.0
assert i == true
assert z == false
```

Strings

Groovy supports:

- Single quotes: ordinary strings.
- Double quotes: ordinary strings with variable expansion (GStrings).
- Triple quotes: multi-line strings with variable expansion (GStrings).
- Slashy strings: strings enclosed in slashes; no need to escape backslashes (useful for regular expressions and file paths).
- operator overloading.

Strings

```
def mlStr = """ I am a multi-line
string!"""
def name = 'Angus';def surname = "Young"
//GStrings
def fullname = "$name $surname"
assert fullname == "Angus Young"
//operator overloading
def s = name * 3
assert s == "AngusAngusAngus"
s = fullname - name
assert s.trim() == surname
//slashy strings: preserve backslashes; no escaping is required
s = /\n\r/
assert s.size() == 4
```

A better syntax for common things

Collections & Maps

```
//Lists (ArrayList behind the scenes)
List<Integer> list = [1, 2, 3]
assert list[0] == 1
//operator overloading
list << 4
assert list.size() == 4
//Maps (LinkedHashMap behind the scenes)
Map<String, String> map = ['one':'1', "two": "2", three:'3']
assert list[1] == map['two'] as Integer
map.one = 'none'
//Java ordinary methods are here
assert map.get('one') == 'none'
list.add(4)
assert list == [1, 2, 3, 4, 4]
//Convert a List to a Set
Set s = list as Set
assert s = [1, 2, 3, 4]
```

Additional Operators

- Safe navigation: `x?.method()`
- Elvis: `x = y ?: "no y"`
- Spread: `["java","groovy"]*.size()` → `[4,6]`
- and more... `<=>` , `=~` , `==~` , `.@` , `.&`

Groovy powerful switch statement

```
switch(val) {
    case "String":   //a string
        break
    case 10..100:    //a range
        break
    case Date:       //a date instance
        break
    case ~/gw+/:     //a reg-ex
        break
    case ['A', 'B']: //a list
        break
    case { it instanceOf Number && it > Integer.MAX_VALUE }
        //a closure
        break
    default:
        //the default, treated as an "else" in Groovy.
}
```

Groovy is Object-Oriented

- Supports Java Interfaces, Classes and Enums.
- Offers additional conventions and facilities (e.g. Groovy properties) that make our life easier.
- Supports Traits: a controlled way to implement multiple inheritance, avoiding the diamond issue.

Everything is an object

Example

```
def x = 1
int y = 1
assert x.class == Integer.class
assert y.class == Integer.class
def l = 1L
assert l.class == Long.class
def z = 0.3
assert z.class == BigDecimal.class
def flag = false
assert flag.class == Boolean.class
```

Numbers

In Groovy all numbers are objects and BigDecimal arithmetic is used by default.

Examples

```
def x = 3
assert x.plus(4) == x + 4
assert x.multiply(4) = x * 4
assert x.mod(4) == x % 4
//BigDecimal arithmetic
assert x/4 == x.div(4)
assert x/4 != x.intdiv(4)
assert 1/2 == 0.5
assert 1/3 == 0.3333333333
```

JavaBeans for human beings

Person.groovy

```
class Person {
    String name
    String surname
}
Person p = new Person(name:"N", surname:"S")
assert p.name == "N"
assert p.getName() == "N"
test.surname = "F"
test.setSurname("F")
```

Closures

Groovy supports functional programming through Closures

👉A closure is an anonymous function together with a referencing environment.

Think of closures as anonymous blocks of code that:

- can accept parameters or return a value,
- can be assigned to variables,
- can be passed as arguments,
- capture the variables of their surrounding lexical scope.

Example

```
//a closure assigned to a variable
def multiplier = { Number x, Number y -> x * y }
//invocation of the closure
assert multiplier(2, 3) == 6
//partial application
def doubler = multiplier.curry(2)
assert doubler(3) == 6
//another doubler
def otherDoubler = { it * 2 } //it -> the first arg
otherDoubler(3) == doubler(3)
```

Collection Manipulation

```
//a groovy list (ArrayList behind the scenes)
def list = [12, 2, 34, 4, 15]
//Collection.collect(Closure c) — extra method available in
    Groovy JDK
//So, given a closure
def inc = { it + 1 }
//we can invoke
list.collect(inc) == [13, 3, 35, 5, 16]
//or pass the closure code directly
assert list.collect{ it + 1 } == [13, 3, 35, 5, 16]
//some more fun
list.findAll{ it > 10 }.groupBy{ it % 2 == 0 ? 'even' : 'odd' }
    == [even:[12, 34], odd:[15]]
```

Owner and delegate

```
class Test {
    long x = 2
    def xTimes = { 1 -> x * 1 }
}
Test test = new Test()
assert test.xTimes.owner == test
assert test.xTimes.delegate == test
test.xTimes(3) == 6
test.x = 3
test.xTimes(3) == 9
def map = [x:4]
assert test.xTimes.resolveStrategy == Closure.OWNER_FIRST
test.xTimes.resolveStrategy = Closure.DELEGATE_FIRST
test.xTimes.delegate = map
assert test.xTimes(3) == 12
```

Meta-programming

Groovy supports meta-proramming

- Categories, Annotations and AST transformations: compile-time metaprogramming.
- Expandos, Meta-classes, Meta-Object protocol: runtime metaprogramming.

Groovy Builders: a powerful concept based on closures and metaprogramming (covered later).

Annotations example

```
@Canonical Person {
    String name
    String surname
}
//@EqualsAndHashCode, @ToString and @TupleConstructor
def p1 = new Person(name:"N", surname:"S")
//but also
def p2 = new Person("N", "S")
//Groovy == is Java's equals()
assert p1 == p2
```

Introduce a new method in Strings

```
String.metaClass.isUpperCase = {
  delegate.toCharArray().every{ Character.isUpperCase(it) }
}
String.metaClass.isLowerCase = {
  !delegate.isUpperCase()
}
assert "JAVA".isUpperCase()
assert "groovy".isLowerCase()
```

Dynamic and Static Features

- In Groovy, most of the type checking is performed at runtime.
- Groovy also supports duck typing.
- Use @TypeChecked & @CompileStatic annotations to maximize static checks during compilation.
- @TypeChecked performs static type checking, yet it dispatches methods through the MOP (permits runtime meta-programming).
- @CompileStatic = @TypeChecked + "static" method linking (no MOP); is the closest you can get to javac-generated bytecode (in terms of behavior & performance).

Duck Typing

Java

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    void onClick(View v) {
        startActivity(intent);
    }
});
```

Groovy

```
button.onClickListener = { startActivity(intent) }
```

Optional typing

Example

```
class Test {
    def s1
    Integer s2 = 2016
    void test() {
        s1 = "NetBeansDay2016"
        assert s1.class == String.class
        s1 = 2016
        s1.class == Integer.class
        s2 = "NetBeansDay2016" //fails at runtime
    }
}
new Test().test()
```

Enable static type checking

Example

```
import groovy.transform.TypeChecked
@TypeChecked class Test {
    def s1
    Integer s2 = 2016
    void test() {
        s1 = "NetBeansDay2016"
        assert s1.class == String.class
        s1 = 2016
        s1.class == Integer.class
        s2 = "NetBeansDay2016" //fails at compile-time
    }
}
new Test().test()
```

Languages and Typing I

A statically-typed language

Resolves the types of variables during compilation » you cannot change the type of a variable.
(Java, opt. Groovy)

A dynamically-typed language

Resolves the types of variables at runtime » you can change the type of a variable.
(Ruby, opt. Groovy)

Languages and Typing II

A strongly-typed language

Guarantees type conformance » you can't coerce a variable to a wrong type.
(Java, Ruby, Groovy)

A weakly-typed language

Has type abstractions that leak » you can screw everything up in all possible ways.
(C)

Seamless Integration with Java

As simple as possible

- You get all groovy magic by just adding a jar in the classpath.
- Call Groovy from Java == call Java from Java.
- Joint compilation: full mix of Java and Groovy.

Mix Java and Groovy I

Fetcher.java

```
1 public interface Fetcher<V> {
2     V fetch();
3 }
```

Person.groovy

```
1 @Canonical class Person implements Fetcher<String>{
2     String name
3     String surname
4     @Override
5     String toString() { "$surname, $name" }
6     @Override
7     String fetch() { toString() }
8 }
```

Mix Java and Groovy II

UsingPerson.java

```
1 public class UsingPerson {
2     public static void main(String[] args) {
3         Person p = new Person("Theodoros", "Kolokotronis");
4         System.out.println(p.fetch());
5     }
6 }
```

Joint compilation

```
$ groovyc Fetcher.java Person.groovy UsingPerson.java -j

$ java -cp groovy-all.jar:. UsingPerson

$ Kolokotronis, Theodoros
```

Groovy Scripts

Groovy can be used as a Java-powered scripting language

Echo.java

```
1 public class Echo {
2     public static void main(String[] args) {
3         if (args != null && args.length > 0) {
4             //for Java < 8 you need a third-party library
5             System.out.println(String.join(" ", args));
6         }
7     }
8 }
```

Echo.groovy (Groovy script – no boilerplate)

```
1 if (args) {
2     println args.join(' ')
3 }
```

A funny example

Java is Groovy is Java

```
public class Echo {
    public static void main(String[] args) {
        if (args != null && args.length > 0) {
            //for Java < 8 you need a third-party library
            System.out.println(String.join(" ", args));
        }
    }
}

$ mv Echo.java Echo.groovy

$ groovy Echo.groovy booh booh

$ booh booh
```

Querying a MySQL database using JDBC

db.groovy

```
@GrabConfig(systemClassLoader=true)
@Grab('mysql:mysql-connector-java:5.1.6')
import groovy.sql.Sql
try {
 def driver = "com.mysql.jdbc.Driver"
 def query  = "select count(*) as c, date(cDate) as d from table
      group by d order by c"
 //Shoot it
 def sql = Sql.newInstance(args[0], args[1], args[2], driver)
 sql.eachRow(query) { row -> println "${row.c}:${row.d}" }
}
catch(e) { e.printStackTrace() }
```

Isn't this Groovy or what?

```
$ groovy db jdbc:mysql://localhost:3306/test test test123
```

Output

```
9427:2004-08-20
6615:2004-10-29
5498:2004-10-08
5103:2004-08-31
4864:2004-10-14
4675:2004-10-31
4583:2004-10-05
4570:2004-08-21
4339:2004-09-30
4235:2004-10-30
```

Part II

Meet Gradle

What is Gradle

Gradle is a software build tool

- Developed by Gradle Inc
- Open source: Apache License v2
- First release 2009, latest release August 2016 (3.0)
- Used by: Android, Spring IO, Linkedin, Netflix, Twitter and more...

At a glance

Gradle

- Combines the best of Ant, Ivy and Maven
  - Cross-platform file management
  - Dependency management
  - Conventions
- In a smart and extensible way
  - Deep API
  - Groovy (and Kotlin in 3.0) DSLs
  - Easy-to-write plugins
- Ultimately offering
  - Clean and elegant builds
  - Plenty of new features
  - Better build management

Features

Non exhaustive list

- Full-fledged programmability
- Both declarative and imperative
- Convention over configuration
- Transitive dependency management
- Multi-project builds
- Polyglot (not only Java)
- Numerous plugins and integrations
- Incremental and parallel execution
- Reporting and analytics
- Embeddable
- Great documentation

A simple Java project

```
apply plugin: "java"
group   = "org.foo.something"
version = "1.0-SNAPSHOT"
repositories {
    mavenCentral()
}
dependencies {
    compile     "commons-io:commons-io:2.4"
    testCompile "junit:junit:4.11"
    runtime     files("lib/foo.jar", "lib/bar.jar")
}
```

The contents of build.gradle (aka the build script), placed in the root folder of the project.

Core Concepts

Build script: a build configuration script supporting one or more projects.

Project: a component that needs to be built. It is made up of one or more tasks.

Task: a distinct step required to perform the build. Each task/step is atomic (either succeeds or fails).

Publication: the artifact produced by the build process.

Dependency Resolution

Dependencies: tasks and projects depending on each other (internal) or on third-party artifacts (external).

Transitive dependencies: the dependencies of a project may themselves have dependencies.

Repositories: the "places" that hold external dependencies (Maven/Ivy repos, local folders).

DAG: the directed acyclic graph of dependencies (what depends on what).

Dependency configurations : named sets (groups) of dependencies (e.g. per task).

Plugins

A plugin applies a set of extensions to the build process.

- Add taks to a project.
- Pre-configure these tasks with reasonable defaults.
- Add dependency configurations.
- Add new properties and methods to existing objects.

Plugins implement the “build-by-convention” principle in a flexible way.

Standard Gradle Plugins

Gradle build language

- Gradle build scripts are Groovy scripts.
- The scripts operate in the context of Gradle's domain objects (DSL objects, aka delegates).
- The main DSL object is a `org.gradle.api.Project` instance.
- There is one-to-one relationship between a Projet object and a build.gradle file.
- Each DSL object exposes its own properties and methods.

Have a look at: Gradle DSL Reference

Back to the example

Clarification

```
//a groovy script with a Project object as the context (delegate
    )
apply plugin: "java"
//invocation of Project.apply(Map) method
group   = "org.foo.something"
version = "1.0-SNAPSHOT"
//update of project.group, project.version properties
repositories {
    mavenCentral()
}
//invocation of the Project.repositories(Closure) method
//From the docs: the closure has a RepositoryHandler object
//as its delegate
dependencies {
    compile     "commons-io:commons-io:2.4"
    testCompile "junit:junit:4.11"
    runtime     files("lib/foo.jar", "lib/bar.jar")
}
//here?
```

A more complex example

```
apply plugin:'java'
//introduce a new task (that invokes a java process)
task generateFiles(type: JavaExec) {
    main = 'some.class.name'
    classpath = sourceSets.main.runtimeClasspath
    args = [ projectDir, 'path/to/gen/files' ]
}
//uses the task 'keyword' (a special case) that
//ends up invoking one of the Project.task() methods
test { //customize the test task
    dependsOn generateFiles
    doLast {
        ant.copy(toDir:'build/test-classes'){
            fileset dir:'path/to/gen/files'
        }
    }
}
//import a class required by this build script
//some necessary classpath definitions are not shown for brevity
import org.apache.commons.io.FileUtils
clean.doFirst { //customize the clean task
    FileUtils.deleteQuietly(new File('path/to/gen/files'))
}
```

Let's build a swing application

Thank you