L1. Agile Cloud Automation

S2. Agile Software Development

Dr A Boronat

Table of Contents

Challenge in Software Development

Goal

- Software release: software that is developed and tested, i.e. our goal
- Build: software that is compiled and assembled (a jar file), intermediate goal to achieve a release

Problems in release management

- Does the code compile?
- Does the code pass the tests? (unit tests)
- Does the code meet the business requirements? (functionality)
- Does the code meet the quality criteria? (performance, security, etc.)

Challenge in Software Development

Solution: continuous delivery

- automatically produce build artifacts (jar files)
- release often and small
- produce a Minimum Viable Product (MVP)
  - to obtain fast feedback from customers
  - reducing risks
  - ensuring continuous progress

How? Agile Methodology

- Iterative, incremental and evolutionary
- People not process (when sensible)
- Focus on quality and on maintaining simplicity
  - continuous delivery: automate as much as possible
    - optimise resources: save time
    - increase quality: to achieve repeatable and consistent processes
  - automated testing
    - quantitative measures
    - consistency
    - release readiness
- Embrace change: very short feedback loop and adaptation cycle

Tooling for Continuous Delivery

Gradle uses Groovy as scripting language to automate builds

Groovy

- Scripting language:
  - no need to declare types
  - facilities for dealing with regular expressions and files
  - versatile syntax
- JVM language: Java-like syntax and Java integration
- In the top 20 of the most popular programming languages:
  - according to TIOBE's index – September'16: #1 Java, #16 Groovy
  - according to TIOBE's index – September'15: #1 Java, #34 Groovy
  - according to RedMonk's ranking – June'16: #2 Java, #20 Groovy
  - according to RedMonk's ranking – June'15: #2 Java, #19 Groovy

Who uses Groovy?

Groovy: syntax

- Java syntax supported (with a few differences):

```
System.out.println("Hello, World!");
```

- but more versatile:

```
println "Hello, World!"
```

- Declaration of variables

```
def var = "Hello, World!"
```

- Strings:
  - single quotes: a string
  - double quotes: string interpolation
  - triple single/double quote: multiline strings

```
def course ='CO7X17'
def text="""
Hello:
$course
"""
println string
```

Groovy: lists and ranges

```
def letters = ['a', 'b', 'c', 'd']
// accessing a member of the list
assert letters[0] == 'a'
// appending
letters << 'e'
// looping a list
for (letter in letters) {
    println letter
}
// ranges
for (number in 1..3) {
    println number
}
```

Groovy: functions

- function declaration

```
def isEven(num) {
  num % 2 == 0
}
```

- function call

```
isEven(2)
```

- function composition

```
def isEven(num) {
  num % 2 == 0
}
def mult2(num) {
  num * 2
}
isEven(mult2(15))
```

Groovy: closures

- block of code that may have parameters

```
{ it -> println it }
{ println it } // it is always included implicitly
```

- calling a closure

```
def printMe = { println it }
printMe 'hello'
```

- closure composition

```
def plus2  = { it + 2 }
def times3 = { it * 3 }
def times3plus2 = times3 >> plus2
times3plus2(5) // result is 17
(times3 >> plus2)(5) // result is 17
```

- closures can be used as parameters

```
[1,2,3].collect({ it + 2 }) // output: a new array [3, 4, 5]
[1,2,3].each({println "Number $it"}) // it may modify the input
[1,2,3,4].find({it % 2 == 0}) // output: 2
[1,2,3,4].findAll({it % 2 == 0}) // output: [2, 4]
[1,2,3,4].any({it % 2 == 0}) // output: true
[1,2,3,4].every({it % 2 == 0}) // output: false
```

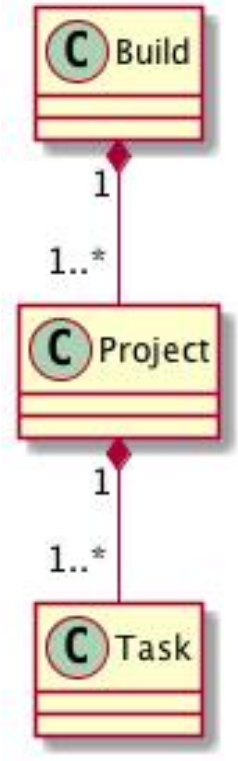
Gradle: a DSL for Build Automation

Features

- Builds atop Ant and Maven
- Provides a DSL to define the build configuration based on Groovy
- Dependency Management
  - Dependencies between projects, to local libraries, to remote repositories
- Build automation
  - Declarative builds and build-by-convention
  - Multi-project builds
  - Scalable

Basic Terminology

- A build consists of one or more projects
- A project is a product to be built or a process to be carried out
  Ex: a library JAR or a web application
  Ex: deploying your application to staging or production environments
- A task is an atomic piece of work which a build performs
  Ex: compiling some classes, creating a JAR, generating Javadoc, or publishing some archives to a repository



Gradle: Basic Tasks

- a task

```
task TaskA
TaskA.description = "task A"
```

- writing actions

```
TaskA.doLast { println "task A" }
TaskA << { println "task A" }
TaskA.doFirst { println "at the start of task A" }
```

- writing tasks as closures

```
task TaskA {
  description "task A"
  doLast { println "taskA" }
}
```

- to execute a task

```
./gradlew TaskA
```

- to show all tasks available

```
./gradlew tasks (--all)
```

Gradle: Tasks Dependencies

- TaskA can execute only if TaskB is executed:

```
task TaskA
task TaskB
TaskA.dependsOn TaskB
```

- equivalently:

```
task TaskA {
  dependsOn TaskB
}
task TaskB
```

- equivalently:

```
task TaskA
task TaskB
TaskB.finalizedBy TaskA
```

Gradle: Properties

- local variables

```
def version = "1.0"
task TaskA {
  description = "task A - version $version"
}
```

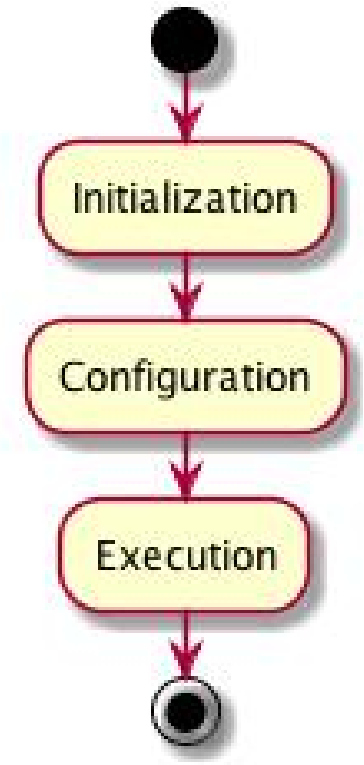
- global variables

```
project.ext.version = "1.0"
task TaskA {
  description = "task A - version $version"
}
```

Build Lifecycle

Initialization

- Determines the projects that will be involved in the build
- Creates a Project instance for each of these projects



Configuration

- Project objects are configured: properties
- Actions are not executed

Execution

- Determines the subset of the tasks to be executed (by the task name arguments passed to the Gradle command and the current directory)
- Gradle then executes the actions in each of the selected tasks.

Gradle: Typed Tasks

- Tasks that are predefined and can be reused: https://docs.gradle.org/current/dsl
- copying files

```
task copyFiles(type: Copy) {
  from 'source'
  into 'target'
}
```

- excluding some files

```
task copyFiles(type: Copy) {
  from 'source'
  into 'target'
  exclude 'file1', 'file2'
}
```

Plugins

A Gradle plugin is an extension to Gradle which configures your project in some way, typically by adding some pre-configured tasks which together do something useful.

Plugins

- Java plugin
  - tasks: compile, unit test, bundle into a JAR file
  - source set: group of source files which are compiled and executed together
    E.g. main, test
  - applying the plugin: `apply plugin: 'java'`
- Eclipse plugin: to generate files that are used by the Eclipse IDE
- Application plugin: to create an executable JVM application
  E.g. java console applications

Project dependencies

Dependencies

- other projects
- external libraries
- internal libraries

Repositories

- where libraries are stored

```
repositories {
  mavenCentral()
}
```

- using dependencies

```
dependencies {
  compile 'group:artifactId:version'
}
```

Goals for this week

TODO list (sprint backlog) on Blackboard:

- Getting your STS ready
- Set up your GitHub repository
- First commit: username
- Groovy: exercises
- Gradle: video and exercises

Groovy exercises

Level of challenge

- ⋆ the solution follows from the resources given (lectures, videos, examples)
- ⋆⋆ the solution may require browsing documentation in order to solve the problem
- ⋆⋆⋆ the solution involves some challenge and it is likely that you will need to combine solutions from different units in order to solve the problem

Exercises

1. dealing with GStrings
2. iterating over collections
3. defining functions
4. using closures
5. using closures and exploring Groovy's API

Gradle exercises

Exercises

1. exercise 1
2. exercise 2

Feedback

Exercises

- solutions to be released next Monday
- laboratory session next Tuesday 9:00-10:00 to discuss any questions you may have about the exercises

I'm stuck...

- **DO NOT** wait until Monday
- **ASK** in the discussion forum on Blackboard:
  - check if your question is related to an existing thread (it may have been answered already)
  - if you can't find anything related to it: create a thread with a meaningful title
  - give context explaining the problem,
  - explain your attempt or your solution, and
  - ask a question

Concluding Remarks

- Agile sw development focusses on delivering working software that is fit for purpose:
  - build automation
  - automated testing
- Gradle provides a DSL for automating release management

http://goo.gl/igM8a1