Notatki - Gradle - Intro

Spis treści

Zakładam, że na etapie czytania tej notatki masz już podstawową wiedzę odnośnie narzędzi do budowania projektów, która została wyniesiona z materiałów o Maven. Dlatego też w przypadku niektórych terminów nie będziemy schodzić głęboko, bo zostały już wyjaśnione wcześniej.

Czym jest Gradle?

Wiemy już do czego jest zdolny Maven. Powiedzmy sobie również co potrafi i czym jest **Gradle**.

Gradle jest narzędziem stosowanym do zarządzania budowaniem aplikacji łącznie z automatyczną obsługą zależności i bibliotek. Obsługuje repozytoria Maven aby te zależności pobierać. Kod piszemy tutaj przy wykorzystaniu **DSL** (Domain Specific Language), którym jest Groovy. Nie musimy całkowicie znać języka Groovy, aby móc tego narzędzia używać. Wiele osób, które stosuje Gradle nie są programistami języka Groovy.

Swoją drogą, wielu osobom też mieszają się te dwie nazwy: Gradle i Groovy. Może żeby było łatwiej to zapamiętać, Gradle - do budowania jak szpadle, Groovy - konfigurację mówi. Albo wymyśl sobie własne rymy 😁.

Zanim jednak przejdę do porównania Maven i Gradle ze sobą, opowiem trochę o tym jak działa Gradle.

Obraz 1. Gradle logo. Źródło: https://commons.wikimedia.org/

Plik z konfiguracją (build script)

Konfiguracja analogiczna do `pom.xml` jest opisywana w plikach `build.gradle` przy wykorzystaniu języka `Groovy`. Jeżeli chcemy uruchomić jakieś polecenie przy wykorzystaniu narzędzia `Gradle`, należy użyć komendy `gradle`.

Projects oraz Tasks

Podczas gdy w `Maven` mówiliśmy o `lifecycle`, `phase` i `goal`, tutaj będziemy mówili o `project` oraz `task`.

Patrząc całościowo, projekt może być reprezentacją końcowego pliku `.jar`, który chcemy wytworzyć, lub końcowej aplikacji webowej, nad którą pracujemy. Projekt może również służyć do reprezentacji końcowego pliku `.zip`. Abstrakcyjne rzecz mówiąc, projekt może być czymś co należy zbudować, lub czymś co należy zrobić. Inaczej mówiąc projekt jest czymś co na koniec chcemy wytworzyć. Aplikacje w praktyce mogą składać się z kilku modułów (czego nie poruszamy), taki moduł jest wtedy projektem w rozumieniu `Gradle`. Do tego projekt składa się z zadań (`task`).

`Task` natomiast, jest to część pracy jaką należy wykonać aby przyczynić się do powstania końcowego `buildu`. Zadaniem może być wykonanie kompilacji kodu, wygenerowanie dokumentacji `Javadoc`, stworzenie pliku `.jar` lub umieszczenie zależności w repozytorium.

Definiowanie własnego taska

Poniżej umieszczam bardzo prosty plik `build.gradle` z własnym taskiem:

```
task ourFirstTask {
    doLast {
        println 'zajavka task'
    }
}
```

Jeżeli teraz chcemy uruchomić powyższe zadanie, należy (w folderze, gdzie mamy zlokalizowany plik `build.gradle`) wykonać komendę:

```
gradle ourFirstTask
```

Na ekranie zostanie wtedy wydrukowane coś podobnego do:

```
Starting a Gradle Daemon, 1 incompatible and 1 stopped Daemons could not be reused, use --status for
details

> Task :ourFirstTask
zajavka task

BUILD SUCCESSFUL in 3s
1 actionable task: 1 executed
```

Jeżeli chcemy ograniczyć się do wydrukowania na ekranie tylko wiadomości, która nas interesuje,

możemy wykonać to samo polecenie z flagą `-q` (quiet). Powoduje to wtedy, że `Gradle` jest mniej "gadatliwy":

```
gradle ourFirstTask -q
```

Jeżeli chcemy przykładowo sprawdzić położenie plików `Gradle` na naszej maszynie, możemy dopisać do tego samego pliku poniższy task i go wywołać:

```
task getHomeDir {
    doLast {
        println gradle.gradleHomeDir
    }
}
```

Taski domyślne

Oprócz tego, że możemy napisać swoje własne taski, możemy również zobaczyć jakie domyślne zadania `Gradle` ma do zaoferowania. W tym celu należy wykonać komendę `gradle tasks -q`. Zobaczysz wtedy na ekranie napisy takie jak **Build Setup tasks** lub **Help tasks**. Są to oznaczenia grupy tasków.

Grupowanie tasków

Jeżeli dołożymy do tego komendę `gradle tasks -q --all` zobaczymy również nasze taski, które nie są przypisane do żadnej grupy. Możemy je przypisać do grupy w ten sposób:

```
task getHomeDir {
    group 'zajavka'
    doLast {
        println gradle.gradleHomeDir
    }
}
```

Jeżeli teraz uruchomimy komendę `gradle tasks -q -all` zobaczymy na ekranie również grupę **zajavka tasks**.

Struktura tasków

Sam task składa się z różnych faz jego wykonania. Możemy wyróżnić fazę konfiguracji, która jest definiowana poza np. fragmentem `doLast`. Dodanie grupy jest przykładem fazy konfiguracji. Oprócz fazy konfiguracji możemy wyróżnić `doFirst` oraz `doLast`. Wiedząc o tym, możemy napisać taką konfigurację:

```
task ourSecondTask {
    group 'zajavka'
    description 'zajavka description'
    println 'Always printed'
}

task ourThirdTask {
    group 'zajavka'
```

```
    doFirst {
        println 'doFirst'
    }

    doLast {
        println 'doLast'
    }
}
```

Jeżeli teraz uruchomimy polecenie:

```
gradle ourThirdTask -q
```

To na ekranie wydrukuje się napis:

```
Always printed
doFirst
doLast
```

Własne taski domyślne

Dołóżmy do tego, że `Gradle` pozwala nam również zdefiniować własne taski domyślne w pliku `build.gradle`, które zostaną uruchomione, jeżeli nie określimy zadania, które ma zostać uruchomione. Przykład poniżej:

```
defaultTasks 'ourFirstTask', 'getHomeDir'

task ourFirstTask {
    group 'zajavka'
    doLast {
        println 'zajavka task'
    }
}

task getHomeDir {
    group 'zajavka'
    doLast {
        println gradle.gradleHomeDir
    }
}
```

Jeżeli teraz uruchomimy komendę:

```
gradle -q
```

Na ekranie zostanie wydrukowane:

```
zajavka task
```

```
C:\Users\karol\.gradle\...\gradle
```

Zależności między taskami

Możemy również określać zależności między zadaniami. Spójrz na konfigurację poniżej:

```
task clean {
    doLast {
        println 'Pretending to clean temporary files'
    }
}

task compile {
    doLast {
        println 'Pretending to compile Java code'
    }
}

task afterCompile(dependsOn: 'compile') {
    doLast {
        println 'Running after compile'
    }
}

task afterClean {
    doLast {
        println 'Running after clean'
    }
}

afterClean.dependsOn clean
```

Spróbuj teraz uruchomić kolejno poniższe polecenia i zobacz jaki będzie rezultat na ekranie:

```
gradle clean -q
gradle compile -q
gradle afterClean -q
gradle afterCompile -q
```

Taski `clean` i `compile` uruchamiają tylko same siebie. `AfterClean` oraz `afterCompile` uruchamiają najpierw odpowiednio `clean` i `compile`, bo tak zostało to zdefiniowane w pliku `build.gradle`.