# Notatki - Gradle - Dependencies

## 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.

## Dependencies

`Gradle` podobnie jak `Maven` wspomaga nas w zarządzaniu zależnościami na naszym `classpath`. Oczywiście tak jak `Maven` - `Gradle` pobiera te zależności za nas. A jak zintegrujemy `Gradle` z Intellij to zależności te mogą być pobierane przy wykorzystaniu przycisku do odświeżenia (tak jak pokazywałem to w przypadku `Maven`).

Znamy już znaczenie terminów takich jak `groupId`, `artifactId` oraz `version`. Zależność w `Gradle` jest definiowana również przy wykorzystaniu tych 3 słów kluczowych. Wygląda to wtedy w ten sposób:

```
group: '<groupId>', name: '<artifactId>', version: '<version>'
```

Lub w wersji skróconej:

```
groupId:artifactId:version
```

Aby dodać zależność, albo zależności do naszego projektu, należy dodać sekcję `dependencies`, przykładowo:

```
plugins {
    id 'java'
}

group = 'pl.zajavka'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'
```

```
repositories {
    mavenCentral()
}

dependencies {
    implementation group: 'org.jsoup', name: 'jsoup', version: '1.14.3'
    // lub w wersji skróconej
    // implementation 'org.jsoup:jsoup:1.14.3'
    testCompile 'junit:junit:4.12'
}

jar {
    manifest {
        attributes 'Main-Class': 'gradle.GradleExamples'
    }
}
```

Pojawiły nam się 3 nowe bloki:

- **plugins** - umożliwia nam używanie kilku pluginów jednocześnie. Zastąpiło fragment `apply plugin: 'java'`.
- **repositories** - określa, że dependencje mają być zaciągane z centralnego repozytorium Maven.
- **dependencies** - blok, w którym możemy definiować zależności, które będą nam potrzebne na różnych `classpathach` (compile vs runtime classpath, pamiętasz?).

Biblioteką `junit` na razie się nie przejmuj. Została ona tutaj dodana żeby pokazać w jaki sposób zapisujemy wiele bibliotek.

# Blok dependencies

W bloku dependencies pojawiły się słówka `implementation` oraz `testCompile`. Są one używane do określenia typów konfiguracji konkretnych dependencji. Upraszczając, określają one w którym momencie dana biblioteka ma być używana na **classpath**. Podstawowe typy konfiguracji, które będziemy stosować (nie wymieniam tu wszystkich możliwych):

- **compileOnly** - zależność jest używana tylko na 'compile classpath'.
- **runtimeOnly** - zależność jest używana tylko na 'runtime classpath'.
- **implementation** - zależność jest używana na obu classpath.
- **testCompileOnly** - zależność jest używana tylko na 'compile classpath', ale tylko w testach.
- **testRuntimeOnly** - zależność jest używana tylko na 'runtime classpath', ale tylko w testach.
- **testImplementation** - zależność jest używana na obu classpath, ale tylko w testach.

Mamy również konfiguracje, które będziesz spotykać w źródłach w internecie, natomiast na moment pisania tego tekstu są one **deprecated** (przestarzałe, kandydat usunięcia w przyszłych wersjach). W nowszych wersjach (np. 7.3) nawet nie możemy ich już zastosować bo są usunięte. Są to:

- **compile** - zamiast tego typu powinniśmy używać `implementation`.
- **runtime** - zamiast tego typu powinniśmy używać `runtimeOnly`.
- **testCompile** - zamiast tego typu powinniśmy używać `testImplementation`.

- **testRuntime** - zamiast tego typu powinniśmy używać `testRuntimeOnly`.

Jeżeli ktoś jest zainteresowany poznaniem większej ilości, umieszczam link.

W dokumentacji `Gradle` można jednocześnie znaleźć informacje dotyczące migracji plików `Gradle` pomiędzy kolejnymi wersjami. Dla zainteresowanych umieszczam link.

# Gdzie Gradle przechowuje pobrane biblioteki?

W przypadku `Maven` mówiliśmy, że pobrane biblioteki są przetrzymywane w katalogu `.m2/repository`. W przypadku `Gradle`, pobrane zależności są przetrzymywane w katalogu `.gradle/caches/modules-2/files-2.1`. Nie są one współdzielone między `Maven` a `Gradle`, są w innych katalogach.

# Blok repositories

Wcześniejszy przykład pokazał jak użyć repozytorium **Maven Central**. Mechanizm repozytoriów został omówiony w materiałach o `Maven`. Oprócz Maven Central możemy również określić swoje własne repozytoria. Przykładowa konfiguracja:

```
repositories {
    maven {
        url "http://repo.zajavka.pl/maven2"
    }
}
```

Możemy również dodać kilka repozytoriów jednocześnie:

```
repositories {
    maven {
        url "http://repo.zajavka.pl/maven2"
    }
    maven {
        url "http://repo.otherwebpage.com/maven2"
    }
}
```

Natomiast możemy w konfiguracji dodać również taki zapis:

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

Nie zostało to pokazane w materiałach, spróbuj usunąć pobraną bibliotekę z katalogu `.gradle/caches/modules-2/files-2.1`, następnie usuń z `gradle.build` wpis dotyczący `mavenCentral()` i zobacz co się stanie z zależnościami `jsoup` w pliku `Main.java`.

## Fat Jar

Z racji, że wcześniej wspominaliśmy o koncepcji `Fat Jar`, spróbujmy napisać konfigurację `Gradle`, która przygotuje **jar-with-dependencies**.

```
plugins {
    id 'java'
}

group = 'pl.zajavka'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'

repositories {
    mavenCentral()
}

dependencies {
    implementation group: 'org.jsoup', name: 'jsoup', version: '1.14.3'
}

jar {
    archivesBaseName = 'myBaseName'
    manifest {
        attributes 'Main-Class': 'gradle.GradleExamples'
    }
    from {
        configurations.runtimeClasspath.collect { it.isDirectory() ? it : zipTree(it) }
    }
}
```

Po dodaniu powyższej konfiguracji i wykonaniu komendy `gradle build`, spróbuj rozpakować plik `.jar` i zobacz jego zawartość.

# Gradle Settings

Oprócz pliku `build.gradle` możemy również stworzyć plik `settings.gradle`. W praktyce jest to używane wtedy gdy nasza aplikacja składa się z kilku modułów. My cały czas obracamy się w zakresie aplikacji, które mają tylko jeden moduł. W takim przypadku, możemy w pliku `settings.gradle` dodać taki wpis:

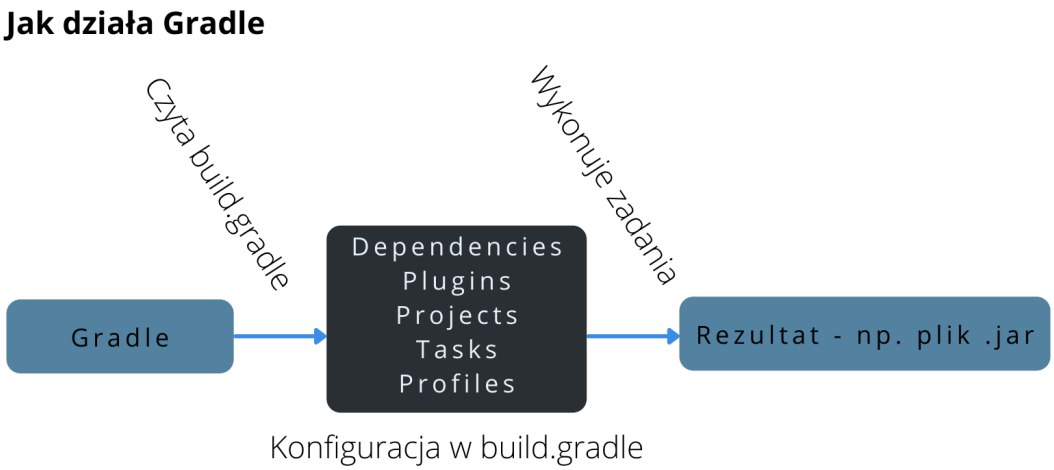
```
rootProject.name = 'java-gradle-examples'
```

Jeżeli natomiast chcielibyśmy zobaczyć z ilu modułów składa się nasza aplikacja, możemy wykonać polecenie:

```
gradle projects
```

# Podsumowanie

Podsumowanie działania `Gradle` umieściłem na grafice poniżej.




*Obraz 1. Jak działa Gradle*