Notatki - Gradle 8

Spis treści

Kompatybilność Gradle i Java

Zaczniemy od tematu, który jest dosyć istotny, gdy zaczniesz zabawę ze zmienianiem wersji Gradle i Javy. Spójrz na tabelkę dostępną tutaj. W tym miejscu znajdziesz informacje o tym, jaka wersja Java wymaga jakiej minimalnej wersji Gradle.

Wiem, że na tym etapie nie omawiamy jeszcze wersji Java i nowości w nich wprowadzanych, ale z racji, że Java jest kompatybilna wstecznie, to możesz pracować u siebie np. z Java 19 (pomimo, że nie poruszyliśmy jeszcze technik, które zostały wprowadzone np. w Java 17) i nadal realizować razem materiały. Oczywiście jakieś drobne zmiany mogą być w kolejnych Javach wprowadzane, przykładowo w Java 19, konstruktor `new Locale` stał się `@Deprecated`. Ale o co w tym chodzi i czym jest `@Deprecated` dowiesz się w dedykowanym warsztacie. Na ten moment istotne jest tylko to, że jeżeli masz zainstalowaną np. Javę 19, to musisz mieć minimalnie Gradle 7.6, co jest zaprezentowane w wymienionej tabelce. Jeżeli zaczniesz coś tutaj kombinować, będziesz mieć błędy, proste ☺.

Gradle 8

Team Gradle wypuścił Gradle 8 w lutym 2023. W jaki sposób można zweryfikować, czy zostały wprowadzone zmiany, które są znaczące (psujące) z naszego punktu widzenia?

Sprawdź istniejące projekty

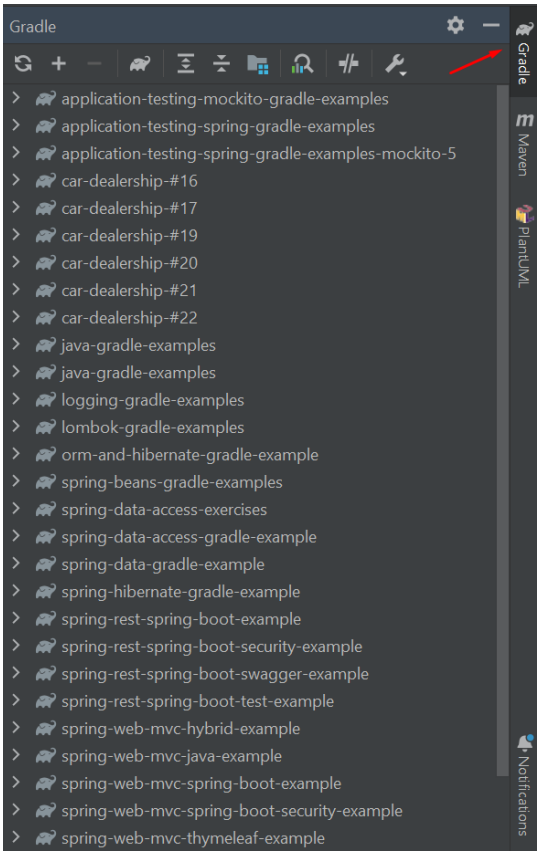
Najlepiej byłoby wziąć jakiś bardziej zaawansowany projekt, który został przygotowany przy wykorzystaniu Gradle, zmienić wersję Gradle w pliku `gradle-wrapper.properties`, spróbować uruchomić `clean` oraz `build` i zobaczyć, czy coś przestało działać.

Plik gradle-wrapper.properties

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
```

```
distributionUrl=https\://services.gradle.org/distributions/gradle-8.0.2-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

Ja (Karol) jestem w trochę bardziej komfortowej sytuacji, gdyż mam przygotowanych bardzo dużo mini projektów z konfiguracjami Gradle, które służą mi do przygotowywania materiałów Zajavka. Ty będziesz zapoznawać się z tymi projektami po kolei w kolejnych warsztatach. Na potwierdzenie, tak wygląda to w moim Intellij:

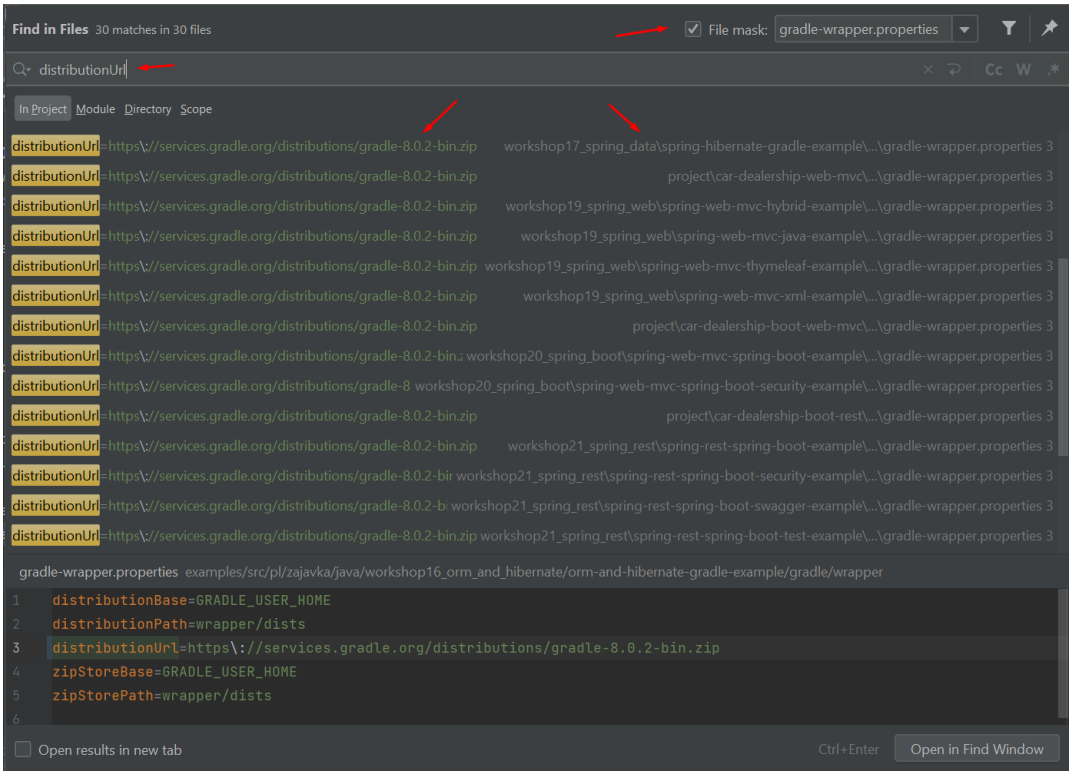


Obraz 1. Intellij Gradle Projects

Zobaczysz tutaj wiele niezrozumiałych jeszcze na tym etapie nazw, ale spokojnie, wszystkie te zagadnienia będą wyjaśniane na przestrzeni kolejnych warsztatów.

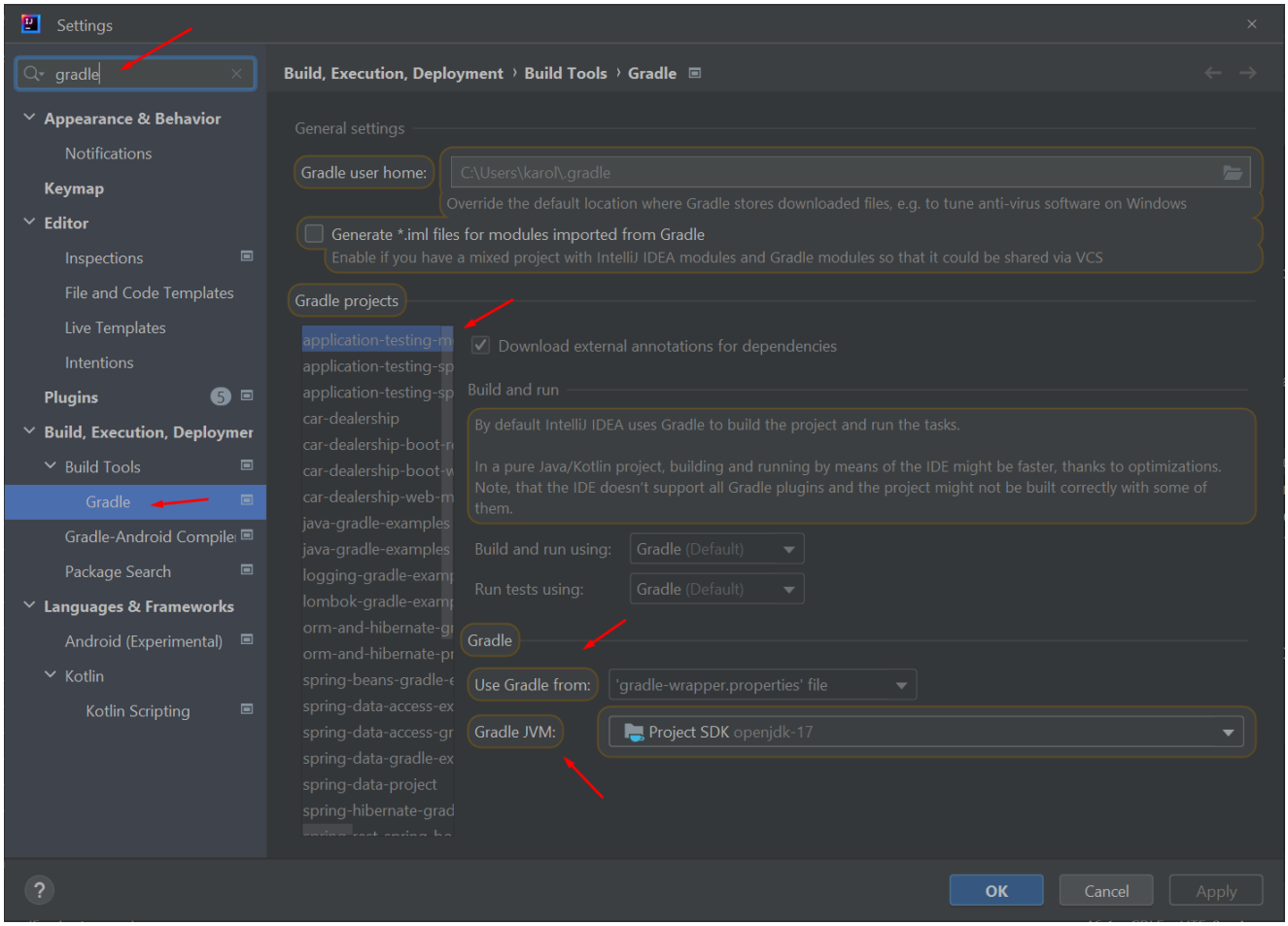
Po co to pokazuję? Żeby pokazać Ci, że testowałem Gradle 8 na wielu z tych konfiguracji Gradle i w żadnej z nich nie wystąpił błąd po zmianie wersji na Gradle 8.

A w jaki sposób to testowałem? Najpierw zamieniłem wersję Gradle we wszystkich plikach `gradle-wrapper.properties`. Każdy z wymienionych projektów ma swój plik `gradle-wrapper.properties`. Wykorzystałem do tego celu skrót *CTRL* + *SHIFT* + *R*. Następnie mogę sprawdzić, wyszukując w całym moim projekcie Intellij, jaka jest ustawiona wartość parametru *distributionUrl*. W tym celu wykorzystuję skrót *CTRL* + *SHIFT* + *F*.



Obraz 2. Intellij Gradle Projects

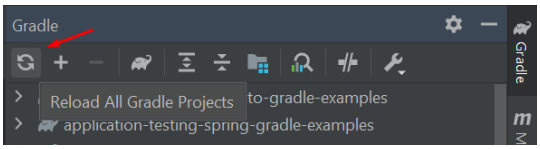
Gdy już zamieniłem wszędzie Gradle Wrapper na wersję 8, upewniłem się jak mam to ustawione w Intellij. W tym celu przeszedłem do ustawień (*CTRL* + *ALT* + *S*) i wyszukałem ustawień Gradle:



Obraz 3. Intellij Gradle Settings

Dla każdego projektu pokazanego tutaj na liście mam ustawione, że *Use Gradle from: 'gradle-wrapper.properties' file*. Zauważ, że ustawiona Java to 17. Przypomnę, że do omówienia zmian w kolejnych wersjach Java przejdziemy w dedykowanym warsztacie.

Teraz przechodzę do zakładki Gradle widocznej po prawej stronie ekranu i wciskam przycisk *Reload All Gradle Projects*:



Obraz 4. Intellij Reload All Gradle Projects

Komputer mi się teraz zaciął tak, że myślałem, że się nie odetnie, ale całe szczęście po ~~tygodniu~~ chwili się udało. Na ekranie nie został wydrukowany żaden błąd.

Co dalej? To samo możemy zrobić, ustawiając Java 19 w ustawieniach Intellij. W moim przypadku projekty nadal działają poprawnie.

W następnej kolejności chcę uruchomić `clean build` dla każdego projektu (czyli rekursywne przeszukiwanie katalogów w celu znalezienia pliku `build.gradle`), który mam zdefiniowany w swoim zajavka repozytorium. W tym celu potrzebuję jakiejś automatyzacji. Chcę zatem wykonać skrypt w Windows CMD, który uruchomi komendę `clean build` i wydrukuje na ekranie rezultat. Do budowania ma zostać wykorzystany Gradle Wrapper (jeżeli taki w projekcie istnieje), jeżeli natomiast nie, ma zostać wykorzystany globalny Gradle.

Przy wykorzystaniu ChatGPT wygenerowałem następujący skrypt:

```
@echo off

echo === PROCESSING START ===

for /r %%d in (.) do (
  pushd "%%d"
  if exist "gradlew.bat" (
    echo === GRADLE WRAPPER FOUND IN: %%d ===
    call gradlew.bat clean build
  ) else if exist "build.gradle" (
    echo === FOUND FILE build.gradle IN: %%d ===
    gradle clean build
  )
  popd
)

echo === PROCESSING END ===
```

Musiałem kilka razy poprawiać ChatGPT albo uściślać wymagania, żeby w końcu osiągnąć efekt, który chcę. Nie udało mi się wygenerować skryptu za pierwszym razem.

Uruchamiam teraz powyższy skrypt w *root* mojego repozytorium, żebym mógł wykonać wszystkie `clean build` i widzę, że wszystko działa poprawnie. To znaczy:

- w późniejszych warsztatach dowiesz się, czym są testy, natomiast testy są uruchamiane przy wykonaniu komendy `clean build`. Testy działały poprawnie z poprzednimi wersjami Gradle i nadal działają poprawnie.
- nie zaobserwowałem żadnego słowa 'ERROR' w informacjach, które są drukowane na ekranie podczas wykonywania skryptu. Są to tak zwane *logi* skryptu. Czym jest logowanie, również dowiesz się w dedykowanym warsztacie.

Po wykonaniu powyższego skryptu można wyszukać na ekranie (lub w pliku, zależy, w jaki sposób uruchomimy komendę) czy pojawiają się takie słowa jak np. *ERROR* albo *FAIL*. W moim przypadku nie pojawiły się żadne błędy, które wynikałyby ze zmiany wersji Gradle, a wszystkie `clean build` zakończyły się sukcesem.

Deprecated Gradle

Przy wykonaniu buildów z tych wszystkich projektów, można natomiast zwrócić uwagę, że na ekranie pojawia się wiadomość:

```
Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come
from your own scripts or plugins.

See https://docs.gradle.org/8.0.2/userguide/command_line_interface.html#sec:command_line_warnings
```

Co możemy zrobić, gdy wystąpi taka sytuacja? W pierwszej kolejności zapoznać się z tą odpowiedzią.

Żeby zrobić to inaczej niż przez flagę `--warning-mode=all`, możemy obok pliku `gradle.build` dodać plik `gradle.properties` i umieścić w nim taki wpis:

Plik gradle.properties

```
org.gradle.warning.mode=all
```

W moim przypadku na ekranie pojawia się wtedy dodatkowy wpis:

```
> Task :compileJava
The GradleVersion.getNextMajor() method has been deprecated. This is scheduled to be removed in Gradle
9.0. Consult the upgrading guide for further information:
https://docs.gradle.org/8.0.2/userguide/upgrading_version_7.html#org_gradle_util_reports_deprecations
```

Task `:compileJava` pochodzi z pluginu *java*, zatem nie jest to błąd, który powodujemy swoim kodem. Możemy ten błąd ewentualnie wyłączyć ustawieniem:

Plik gradle.properties

```
org.gradle.warning.mode=none
```

A jakieś nowe rzeczy?

Rozczaruję Cię, ale w Gradle 8 nie zostało wprowadzone nic nowego, co jest istotne na tym etapie nauki z naszego punktu widzenia. Dlatego właśnie chciałem poświęcić czas i pokazać Ci, że projekty, które miałem przygotowane w Gradle 7, nadal działają poprawnie.

A jak nie będzie działać?

W przyszłości może wystąpić taka sytuacja, że konfiguracja napisana w Gradle 7 z jakiegoś powodu przestanie działać, gdy zrobimy update np. do Gradle 10. Co wtedy zrobić? Znaleźć taki Migration Guide i poszukać, co uległo zmianie. Na tej podstawie ocenić, co u nas nie działa i co należy poprawić.

Podsumowanie

W programowaniu to jest normalne (można powiedzieć, że jest to chleb codzienny dewelopera), że wersje narzędzi i bibliotek ulegają zmianie. Ciągnie to za sobą takie konsekwencje, że deweloper musi na bieżąco aktualizować wersje narzędzi i bibliotek w swoich projektach i śledzić, że takie zmiany w ogóle występują.

Oczywiście istnieją narzędzia, które umożliwiają taką automatyczną aktualizację wersji i przykładowo, jeżeli będziesz w przyszłości korzystać z GitHub, to dostawca ten posiada rozwiązanie nazwane Dependabot, które służy do automatyzacji aktualizowania wersji zależności.

Spokojnie, jeżeli chodzi o tego GitHuba. Wrócimy do niego w dedykowanym warsztacie, wtedy dowiesz się, co to jest i po co to jest. Chciałem tutaj tylko zaznaczyć, że są narzędzia do automatyzowania podbijania wersji bibliotek i podałem przykład.

Release Notes

Z każdym nowym wdrożeniem jakiegoś narzędzia, twórcy najczęściej przygotowują coś takiego jak Release Notes, czyli informacje o tym, *co uległo zmianie*. W przypadku Gradle 8 możesz takie coś znaleźć tutaj. Z tego miejsca dowiesz się jakie zmiany zostały wprowadzone i jakie błędy zostały naprawione. Błędy znajdziesz na dole w sekcji *Fixed issues*.

To na podstawie takich notatek (Release Notes) oraz Migration Guides, deweloperzy, pracując z dokumentacją, dostosowują swój projekt do nowszych wersji narzędzi, o ile taka konieczność dostosowania w ogóle wystąpi. W naszym przypadku nie wystąpiła, dlatego podałem tylko źródła i wskazówki, na co należy zwracać uwagę.

Filozofia dotycząca kursu

Na koniec tego wpisu chcę zaznaczyć pewną istotną kwestię dotyczącą organizacji materiałów w tym kursie. W kolejnych warsztatach będziemy korzystać w ogromnej większości z Gradle. W notatkach będą się pojawiały przykłady analogiczne napisane w Maven, ale na filmach będziemy korzystać z Gradle, żeby móc stosować krótszą formę zapisu i więcej zmieścić na ekranie ☺. W ramach ćwiczeń zachęcam jednak do pisania tego samego w Gradle i w Maven.

Materiały dotyczące Hibernate, czy Spring (m.in. o tym będziemy rozmawiać w kolejnych warsztatach) zostały przygotowane przy wykorzystaniu Gradle 7. Oznacza to, że w kolejnych warsztatach, za każdym razem, gdy będziemy dodawać kolejne nowe rzeczy, będą one realizowane w Gradle 7. Gradle będzie nam służył jako narzędzie do organizacji projektu i jego sposób działania (oraz sposób myślenia o nim) nie ulegają zmianie z kolejnymi wersjami tego narzędzia.

Z racji, że w tym kursie zależy nam na przedstawieniu Ci sposobu myślenia o konkretnych mechanizmach i narzędziach, będziemy reagować na zmiany wersji narzędzi wtedy, gdy będą one wprowadzały istotne zmiany. Co mam na myśli? Kolejne materiały zostały przygotowane przy wykorzystaniu Gradle 7, ale możesz je również realizować, korzystając z Gradle 8 i wyższych wersji. Jeżeli w kolejnych wersjach będzie pojawiało się coś istotnego, będziemy to dodawać do kursu na odpowiednich etapach.

Czyli jeżeli będziesz oglądać materiały z Gradle 7 i coś Ci nie będzie działać, to spokojnie, na pewnym etapie do tego wrócimy, a Ty albo możesz spróbować rozwiązać problem przy wykorzystaniu wszystkich dostępnych możliwości, albo poczekać do momentu, aż dodamy materiał pokazujący jak radzić sobie z tym problemem.

To samo występuje z resztą w przypadku "czystej" Javy. Bootcamp Zajavka to podstawy programowania w Java, natomiast omówienie funkcjonalności wprowadzanych w kolejnych wersjach Javy następuje dopiero w dedykowanym warsztacie.

Czyli co? Po prostu nam zaufaj i ucz się jak Ci wygodnie. Jak coś nie działa, jak powinno, korzystaj z Internetu, albo rób na takich wersjach, jak pokazujemy ☺.