Notatki - Gradle - Java

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.

Gradle wrapper

Jeżeli chcielibyśmy móc uruchamiać zadania `Gradle` nie mając zainstalowanego `Gradle` lokalnie, możemy wykorzystać **Gradle** `wrapper`. W momencie, gdy uruchamiamy `Gradle` stosując **wrapper**, konkretna wersja `Gradle` jest automatycznie pobierana i używana do wykonania danego builda. Pozwala to na niezależność od zainstalowanej wersji `Gradle` na maszynie, na której pracujemy. Załóżmy, że mamy zainstalowanego `Gradle` w wersji 6.8 i potrzebujemy uruchomić build dla 3 projektów, które stosują inne wersje `Gradle`. Dzięki `Gradle` **wrapper** możemy wykonać buildy z różnymi wersjami `Gradle` w różnych projektach. Aby faktycznie wygenerować pliki wrappera, musimy mieć zainstalowany `Gradle` na naszym systemie operacyjnym.

Aby wygenerować pliki wrappera można zastosować poniższą komendę. Możemy też nie podawać wersji, `Gradle` weźmie wtedy najnowszą.

```
gradle wrapper --gradle-version <wersja>
```

Zamiast `<wersja>` możemy wpisać np. `7.4` lub `7.6`.

Wygenerowane wtedy zostaną pliki:

```
<project folder>
    gradlew
    gradlew.bat
    gradle
        wrapper
            gradle-wrapper.jar
            gradle-wrapper.properties
```

Uruchamiamy wtedy taski `Gradle` przy wykorzystaniu polecenia:

```
./gradlew build
// lub jak nam nie zadziała, to zwyczajnie
gradlew build
```

Gradle Wrapper jest świetną możliwością, która pozwala nam dostosowywać wersję Gradle do konkretnego projektu. Jeżeli masz zainstalowany Gradle globalnie (na cały system operacyjny) w wersji np. 7.3, to możesz w trzech różnych projektach korzystać z różnych wersji Gradle np. 7.4, 7.5 oraz 7.6. Wszystko jest kwestią wartości ustawionej w pliku `gradle-wrapper.properties`. Plik ten jest wtedy dedykowany dla konkretnego projektu.

Plik gradle-wrapper.properties

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-7.6-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

Budowanie projektów Java

`Gradle` zawiera wbudowany domyślnie plugin umożliwiający kompilację kodu źródłowego Javy, uruchomienie testów (o tym później), tworzenie plików `.jar` oraz generowanie dokumentacji `Javadoc`. Ponownie odwołam się to terminu **convention over configuration**. W tym przypadku stosowanie konwencji w projekcie budowanym przy pomocy `Gradle` oznacza zachowanie określonej struktury projektu. Plugin Java zakłada istnienie folderów:

- **src/main/java** - w którym umieszczamy pliki źródłowe aplikacji Java
- **src/test/java** - w którym umieszczamy pliki źródłowe **testów** naszej aplikacji

Jeżeli będziemy trzymać się tej konwencji, jedyne co jest nam potrzebne aby określić konfigurację do budowania projektu Java to stworzenie pliku `build.gradle` z poniższą linijką:

```
apply plugin: 'java'
```

Więcej o możliwościach tego pluginu można poczytać tu.

Po stworzeniu konfiguracji zawierającej tę linijkę, uruchom komendę `gradle tasks`. Możesz zwrócić uwagę, że zostało dodanych dużo nowych tasków, które są specyficzne dla Javy. Dzięki temu będziemy teraz mogli wykorzystać task służący do budowania. Aby w tym momencie uruchomić build, należy wywołać komendę `gradle build`.

Task `gradle build` kompiluje kod, wykonuje testy (o których powiemy później) oraz tworzy wynikowy plik `.jar`. Aby na tym etapie zobaczyć rezultaty wykonanej komendy należy spojrzeć do folderu `build`. W środku znajdziemy następujące katalogi:

- **classes** - w środku znajdziemy skompilowane pliki `.class`. Tutaj należy szukać plików `.class`

odpowiadających naszym plikom źródłowym z katalogu `src/main/java`

- **reports** - w środku znajdziemy wygenerowane raporty, które są tworzone np. przy uruchamianiu testów naszego programu. Na ten moment jednak nie dodaliśmy żadnych testów do projektu
- **libs** - w środku znajdziemy pliki `.jar` utworzone w trakcie trwania buildu. Zwróć uwagę, że utworzony plik `.jar` nie zawiera wersji

Wersja pliku jar

Jeżeli chcemy określić wersję pliku `.jar`, który wygenerowaliśmy, należy dodać do pliku `build.gradle` następującą linijkę:

```
apply plugin: 'java'
version = '0.1.0'
```

Wersja Javy

Tak jak w przypadku `Maven`, mogliśmy określić parametry takie ja `source` i `target` służące do podania kompatybilności z wersją Javy odpowiednio na etapie kompilacji lub uruchamiania programu, tak samo możemy zrobić to tutaj. Do pliku `build.gradle` możemy dodać parametry takie jak:

- **sourceCompatibility** - cytując dokumentację - Java version compatibility to use when compiling Java source. Default value: version of the current JVM in use.
- **targetCompatibility** - cytując dokumentację - Java version to generate classes for. Default value: sourceCompatibility.

```
apply plugin: 'java'
version = '0.1.0'
sourceCompatibility = '17'
```

Skoro dokumentacja mówi, że wartość domyślna parametru `targetCompatibility` to `sourceCompatibility` to ustawiamy tylko wartość parametru `sourceCompatibility`.

Jeżeli natomiast chcemy zobaczyć która wersja `Gradle` wspiera którą wersję Javy, można to zrobić w dokumentacji.

Executable jar

Co zrobić jeżeli chcemy aby nasz plik `.jar` był uruchamialny? Czyli chcemy mieć możliwość uruchomienia naszego programu bezpośrednio z pliku `.jar`. W paczce `gradle` w katalogu `src/main/java` stwórzmy poniższy plik:

```
package pl.zajavka;

public class GradleExamples {

    public static void main(String[] args) {
```

```
        System.out.println("Gradle examples");
    }
}
```

Do tego w pliku `build.gradle` określmy poniższą konfigurację:

```
apply plugin: 'java'

jar {
    archivesBaseName = 'myBaseName'
    manifest {
        attributes 'Main-Class': 'pl.zajavka.GradleExamples'
    }
}
```

Jeżeli teraz uruchomimy komendę `gradle clean build`, to możemy spróbować uruchomić nasz program `.jar` za pomocą komendy `java -jar <nazwa_pliku_jar>.jar`.

Taski w 'gradle build'

Na stronie z dokumentacją pluginu o tutaj znajduje się obrazek o ten przedstawiający jakie zadania są wykonywane przed zadaniem `build`, w ramach zależności między taskami. Pokazywaliśmy to wcześniej. Jeżeli spojrzymy pod ten adres zobaczymy opis tych zadań i ich wzajemne zależności. Na tej podstawie widać, że zanim wykonany zostanie task `build`, wcześniej muszą się wykonać też inne, takie jak chociażby `check` i `assembly`.