Notatki - Lombok - Intro

Spis treści

Czym jest lombok?

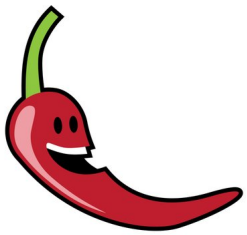
`Lombok` jest jedną z bibliotek, która ma za zadanie ułatwić nam pracę. Każda biblioteka rozwiązuje jakiś problem. Tak samo jest w tym przypadku.

Przypomnijmy, że biblioteka (library) to zestaw funkcji/metod, które można wywołać, które są zorganizowane w postaci klas. Inaczej sprawę ujmując są to gotowe klasy oraz metody, które możemy wykorzystywać w swoim kodzie/programie, które zostały napisane przez kogoś innego.

Java w swojej naturze jest **gadatliwa** (verbose). Stwierdzenia tego używa się w praktyce w stosunku do sytuacji, gdzie trzeba napisać dużo kodu bądź instrukcji aby osiągnąć jakiś cel. Jednocześnie też taki kod lub takie komendy nie przynoszą dużo wartości. Wartość w tym rozumieniu to kod, który służy do realizowania funkcji biznesowych aplikacji. W Javie poznaliśmy już dużo kodu, który "nie realizuje biznesu". Przykładem takiego kodu są konstruktory (takie proste nie realizujące nic innego jak przypisanie), gettery, settery, equals(), hashCode() lub toString().

Aby rozwiązać ten problem powstało coś takiego jak Project Lombok. Działa to w ten sposób, że zamiast pisać tego rodzaju kod, dołączamy bibliotekę, która na podstawie adnotacji (przykład adnotacji to `@Override`) określa, że w danej klasie mamy mieć konstruktor, getter itp. Wtedy w trakcie procesu budowania programu, na podstawie tych adnotacji automatycznie jest generowany bytecode w plikach `.class`, który zawiera wspomniane konstruktory, gettery itp. Inaczej mówiąc, nie musimy tworzyć tych konstruktorów ręcznie, wystarczy, że dodamy odpowiednie adnotacje w pliku `.java`.

Przypomnijmy, że programy w praktyce pisze się po to, żeby ktoś mógł dzięki nim szybciej bądź taniej zarabiać pieniądze. Tak samo jak automatyzacja fabryk lub linii produkcyjnych. Im szybciej możemy napisać faktyczny kod realizujący "biznes" tym lepiej, szybciej osiągamy finalny cel tworzenia takiego oprogramowania.



Obraz 1. Lombok logo. Źródło: https://medium.com/

Jak skonfigurować Lombok w projekcie?

Gradle

Jeżeli korzystamy z Gradle, tutaj znajdziemy instrukcję konfiguracji. Poniżej umieszczam gotowy plik `build.gradle`.

```
plugins {
    id 'java'
}

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

repositories {
    mavenCentral()
}

dependencies {
    compileOnly 'org.projectlombok:lombok:1.18.22'
    annotationProcessor 'org.projectlombok:lombok:1.18.22'

    testCompileOnly 'org.projectlombok:lombok:1.18.22'
    testAnnotationProcessor 'org.projectlombok:lombok:1.18.22'
}
```

Jak już się domyślasz, dopisek `test`, znaczy, że zależność będzie używana w testach. My na razie nie rozmawiamy o testach, więc ten fragment pominiemy (może zostać w `build.gradle`, przyda się na potem).

Wyjaśnienie czym jest `annotationProcessor` nastąpi później.

Maven

Jeżeli korzystamy z Maven, możemy skorzystać z konfiguracji opisanej w tym źródle. Korzystając ze wspomnianego źródła, możemy napisać taką konfigurację:

```
<project>
    <modelVersion>4.0.0</modelVersion>

    <groupId>pl.zajavka</groupId>
    <artifactId>lombok-maven-examples</artifactId>
```

```
    <version>1.0.0</version>

    <properties>
        <java.version>17</java.version>
        <lombok.version>1.18.22</lombok.version>
        <maven-compiler-plugin.version>3.8.0</maven-compiler-plugin.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>${lombok.version}</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven-compiler-plugin.version}</version>
                <configuration>
                    <release>${java.version}</release>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                            <version>${lombok.version}</version>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Tag `scope` i jego wartość wynikają z tego, że potrzebujemy tej zależności na etapie kompilacji, aby stworzyć pliki `.class` z wygenerowanym kodem. `Lombok` nie jest już natomiast potrzebny w trakcie działania programu.

Dokumentacja `Lombok` wspomina również, o dodaniu `annotationProcessorPaths` w przypadku wersji Javy wyższej niż 9.

Annotation Processor

Co oznaczają słówka `annotationProcessor` oraz `testAnnotationProcessor` w przypadku **Gradle** oraz analogicznie w przypadku **Maven**?

Annotation processing jest narzędziem wbudowanym w `javac` do skanowania i przetwarzania adnotacji w trakcie trwania kompilacji. Możemy zarejestrować własny procesor adnotacji. Procesor taki jest w stanie na podstawie zdefiniowanych adnotacji generować kod na wyjściu. Wspomniałem wcześniej, że `Lombok` daje nam możliwość generowania kodu na podstawie zdefiniowanych adnotacji, dlatego też w ogóle rozmawiamy o **Annotation processing**.

Jeżeli chcemy taki procesor zarejestrować, to możemy to zrobić na 2 sposoby. Ręcznie, albo przez narzędzie.

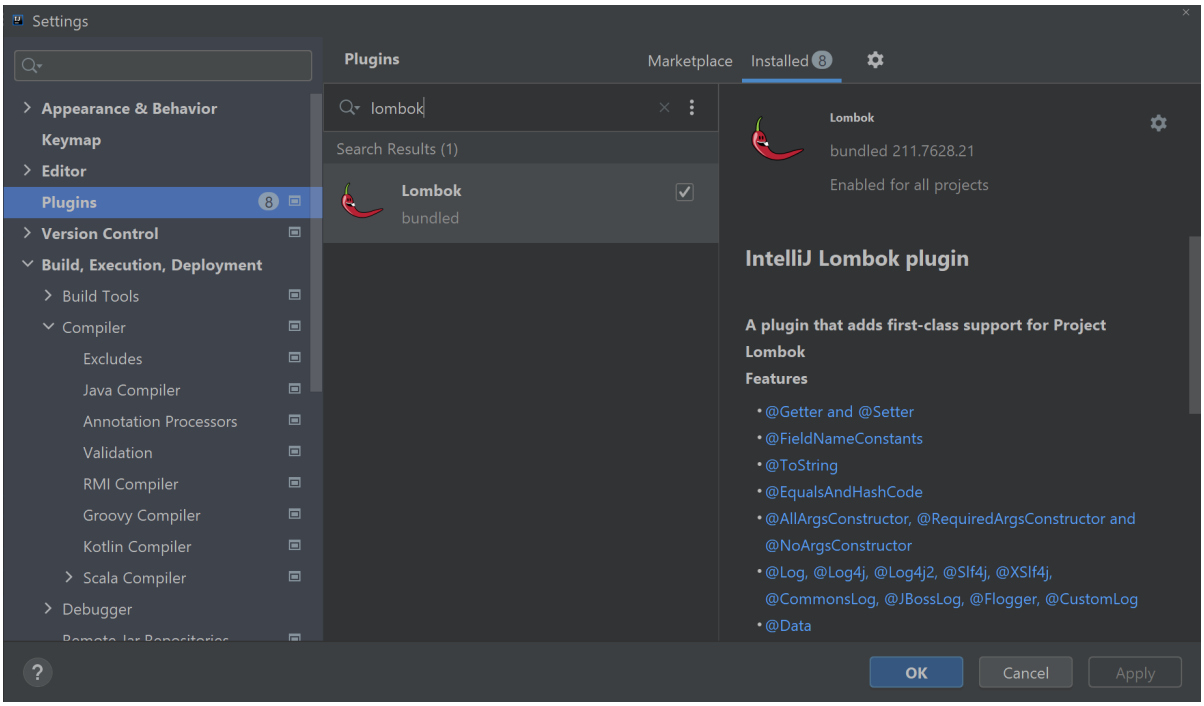
Jeżeli chcemy zrobić to ręcznie, należy dodać opcję `--processor-path path` lub `-processorpath path` do wywołania komendy `javac`. Cytując dokumentację:

> `--processor-path path` or `-processorpath path`
>
> Specifies where to find annotation processors. If this option is not used, then the class path is searched for processors.

Jeżeli natomiast chodzi o rejestrację takiego procesora, dokumentacja dedykowana pod **Maven** wspominała o ręcznej konfiguracji takiego procesora. Wspomina o tym również dokumentacja dedykowana pod **Gradle**. Na wszelki wypadek, przed konfiguracją `Lombok` upewnij się co mówi dokumentacja.

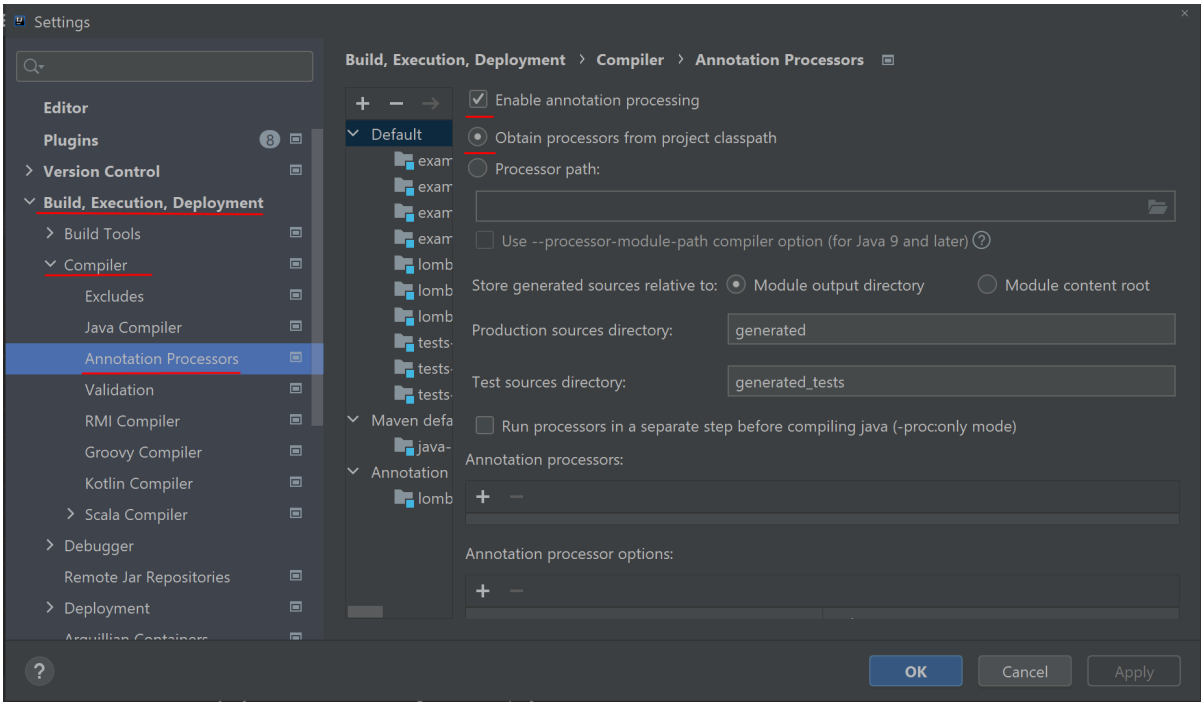
Intellij

Do tego należy dodać plugin `Lombok` do Intellij (*CTRL* + *ALT* + *S* > *Plugins*). Jeżeli nie będziemy mieli dodanego pluginu `Lombok`, Intellij będzie pokazywał nam błędy kompilacji, bo np. nie będzie widział danego konstruktora, gettera itp.



Obraz 2. Lombok Intellij Plugin

Jeżeli nadal coś nie będzie działało, można zobaczyć ustawienia, które pokazuję poniżej:



Obraz 3. Lombok Intellij Settings

Definicje

POJO

POJO czyli **P**lain **O**ld **J**ava **O**bject. Oznacza klasę, która w żaden sposób nie jest zależna od zewnętrznych bibliotek. Nie oznacza to stosowania żadnej konwencji, tzn, posiadania getterów, setterów itp. Ważne jest to, że klasa taka może być używana beż zależności do jakichkolwiek bibliotek czy frameworków.

Framework można na tym etapie rozumieć jak dużą bibliotekę, która w uproszczeniu może służyć jako szkielet aplikacji.

Java Bean

Czytając o przeróżnych aspektach, napotkasz w końcu stwierdzenie `Java Bean`. Określenie to narzuca pewnego rodzaju standard klasy Java, w której:

- Wszystkie pola są prywatne, a dostęp do nich jest realizowany przez gettery i settery
- Nazewnictwo getterów i setterów trzyma się konwencji `getX()` i `setX()`
- Mamy dostępny publiczny bezargumentowy konstruktor
- Klasa ta implementuje interfejs `Serializable`

Nazwa **Bean** została nadana, aby uwypuklić ten standard, którego celem jest tworzenie reużywalnych komponentów (klas). Reużywalnych w tym rozumieniu, że działanie bibliotek możemy wtedy oprzeć o obiekty z naszego projektu pod warunkiem, że spełniają one standardy. Można się też pokusić o stwierdzenie, że Intellij wspiera ten standard, dając nam możliwość generowania getterów, setterów i konstruktorów.

Boilerplate

Cytując Wikipedię:

> In computer programming, boilerplate code or just boilerplate are sections of code that are repeated in multiple places with little to no variation. When using languages that are considered verbose, the programmer must write a lot of code to accomplish only minor functionality. Such code is called boilerplate.

Czyli jest to taki kod, o którym wspominałem wcześniej. Proste konstruktory, gettery, settery, toString itp.