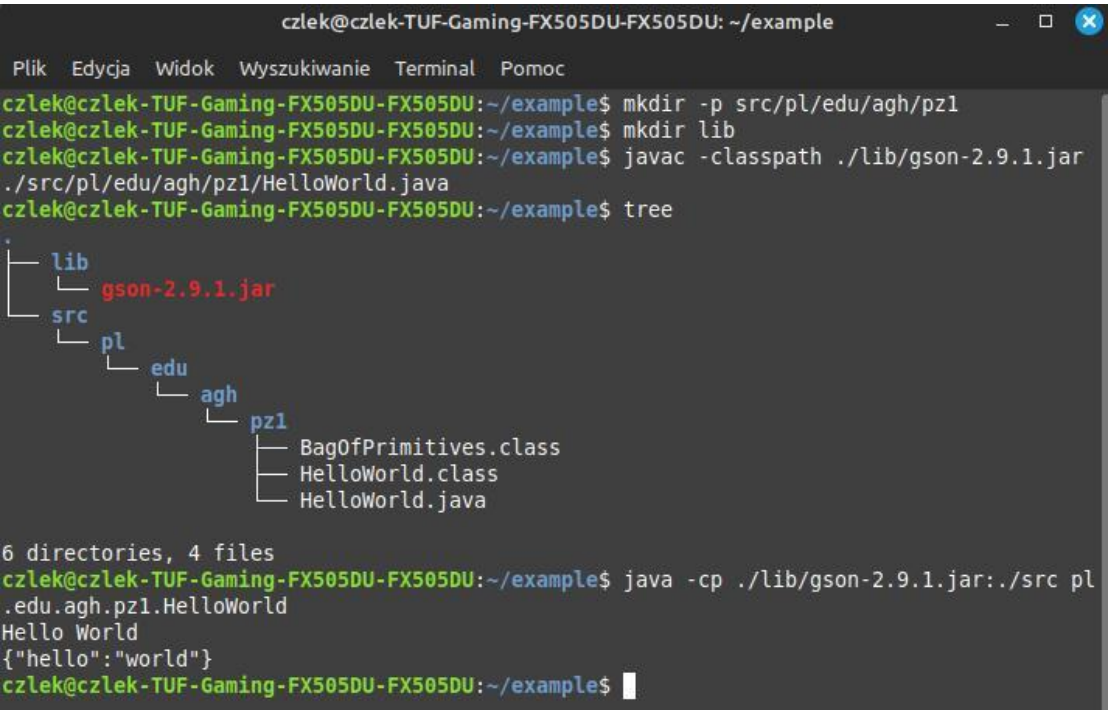
Apache Maven

Narzędzie do zarządzania budowaniem, testowaniem i dokumentacją projektu.

Cele Mavena:

- uczynienie procesu budowania projektu prostszym
- zapewnienie standaryzacji w ramach systemu budowania
- dostarczenie wartościowych informacji o projekcie
- promowanie dobrych praktyk deweloperskich.

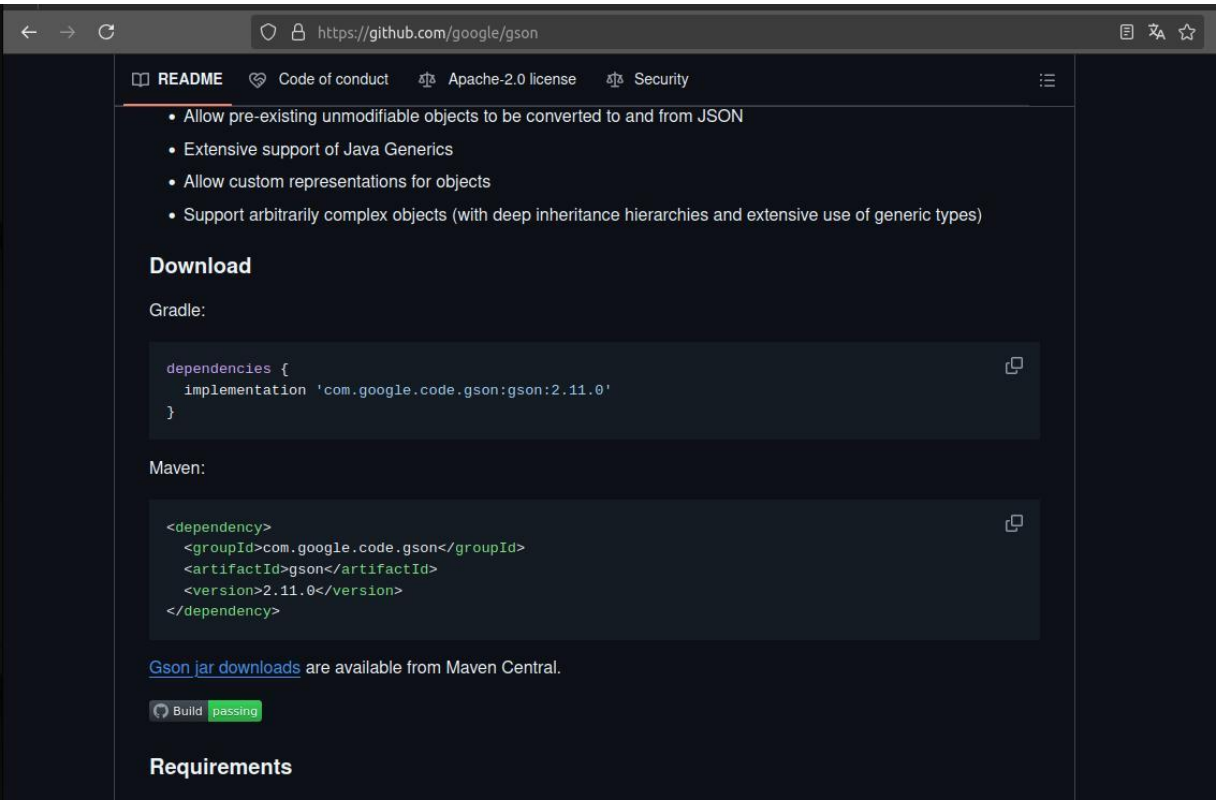
Jak bez Mavena?



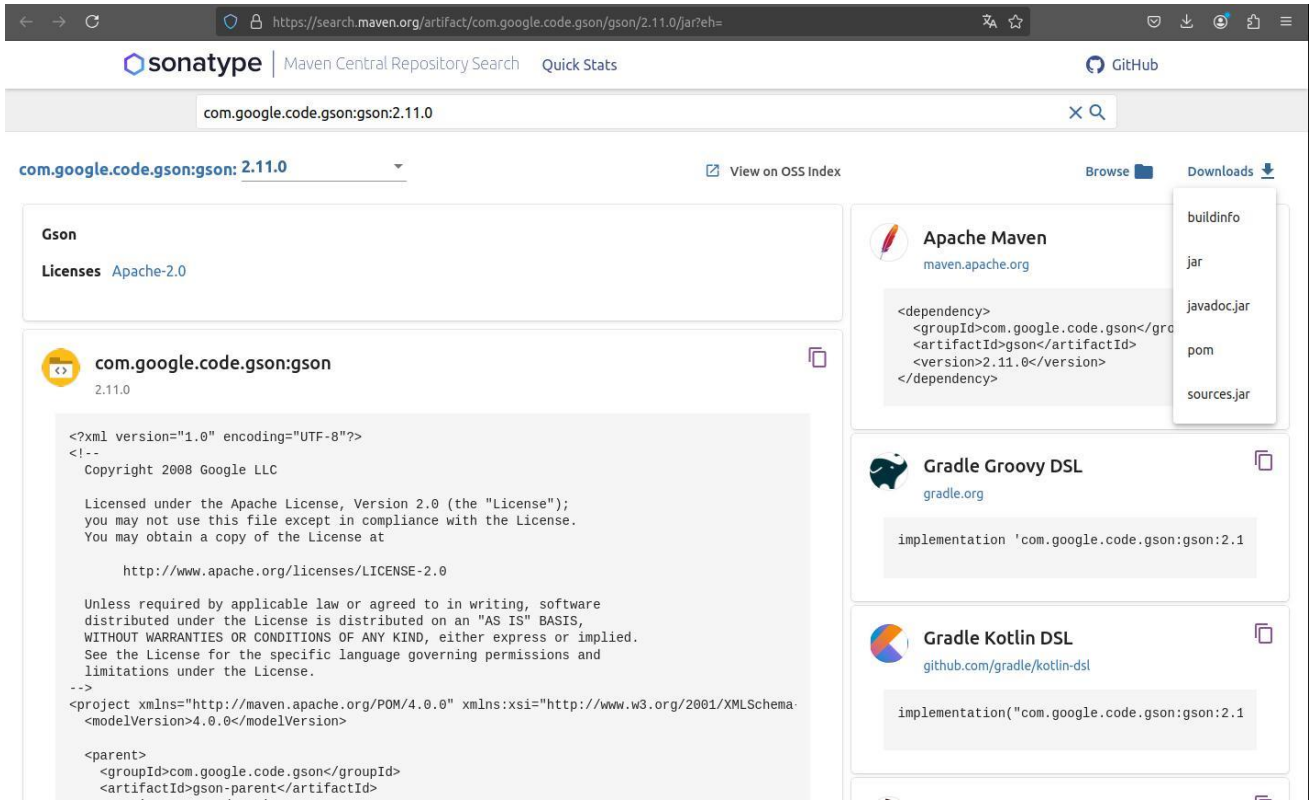
HelloWorld.java

```java
package pl.edu.agh.pz1;

import com.google.gson.Gson;

class BagOfPrimitives {

  private String hello = "world";
  BagOfPrimitives() {
    // no-args constructor
  }
}

public class HelloWorld {
    public static void main(String ... args) {
        System.out.println("Hello World");
        BagOfPrimitives obj = new BagOfPrimitives();
        Gson gson = new Gson();
        String json = gson.toJson(obj);
        System.out.println(json);
    }
}
```

Jak bez Mavena?



repozytorium Gsona



repozytorium Mavena

Instalacja

Pobranie pliku binarnego z oficjalnej strony (https://maven.apache.org/download.cgi) i rozpakowanie:

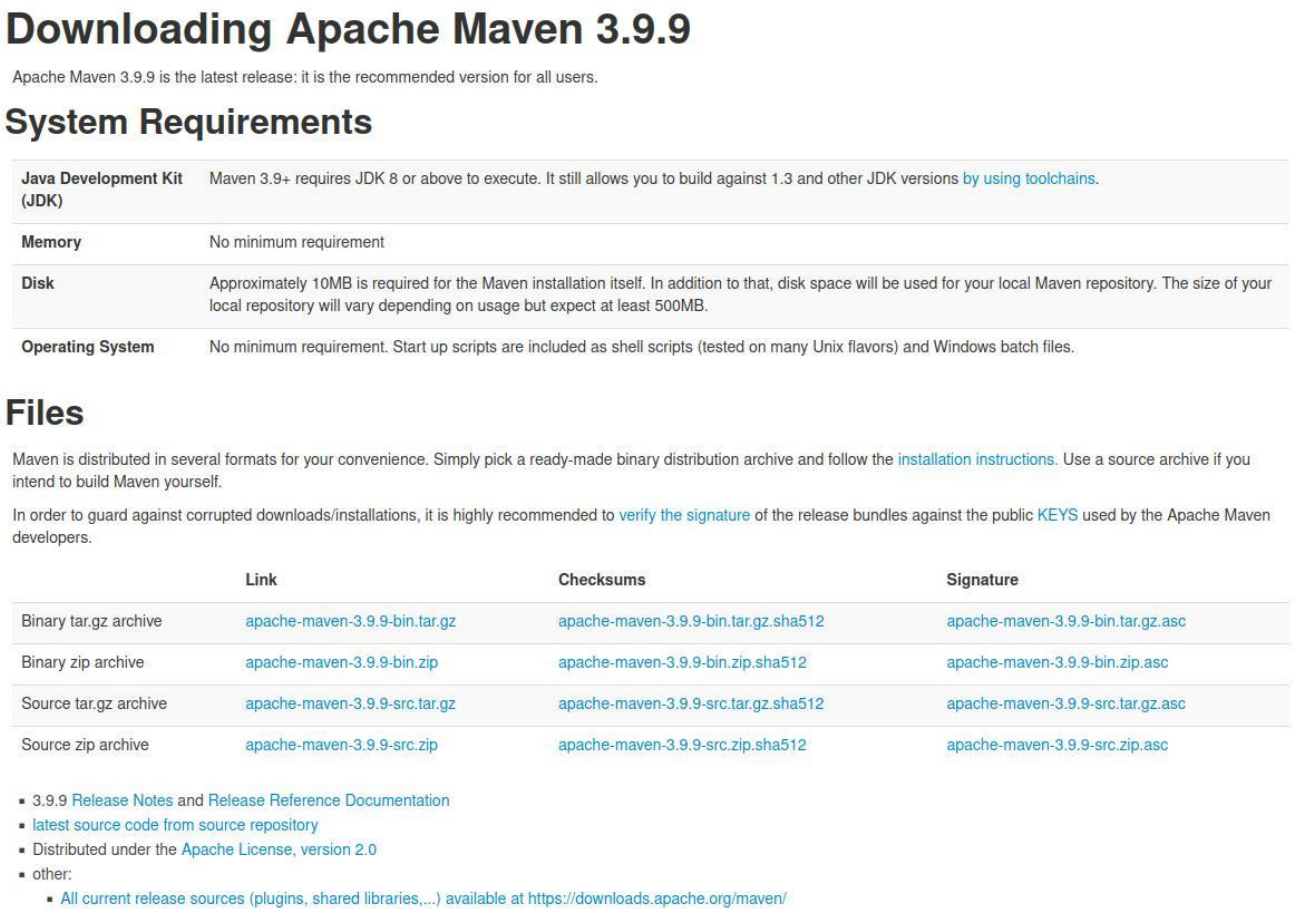
```
$ unzip apache-maven-3.9.9-bin.zip
lub
$ tar xzvf apache-maven-3.9.9-bin.tar.gz
```

oraz dodanie folderu bin do $PATH np.

```
$ echo 'PATH="$PATH:/path_to_file/bin"'
> ~/.bashrc
```

albo użycie własnego menadżera pakietów, np.:

```
$ sudo apt install maven
```



Downloading Apache Maven 3.9.9

Apache Maven 3.9.9 is the latest release: it is the recommended version for all users.

System Requirements

Java Development Kit (JDK)	Maven 3.9+ requires JDK 8 or above to execute. It still allows you to build against 1.3 and other JDK versions by using toolchains.
Memory	No minimum requirement
Disk	Approximately 10MB is required for the Maven installation itself. In addition to that, disk space will be used for your local Maven repository. The size of your local repository will vary depending on usage but expect at least 500MB.
Operating System	No minimum requirement. Start up scripts are included as shell scripts (tested on many Unix flavors) and Windows batch files.

Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the installation instructions. Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.9.9-bin.tar.gz	apache-maven-3.9.9-bin.tar.gz.sha512	apache-maven-3.9.9-bin.tar.gz.asc
Binary zip archive	apache-maven-3.9.9-bin.zip	apache-maven-3.9.9-bin.zip.sha512	apache-maven-3.9.9-bin.zip.asc
Source tar.gz archive	apache-maven-3.9.9-src.tar.gz	apache-maven-3.9.9-src.tar.gz.sha512	apache-maven-3.9.9-src.tar.gz.asc
Source zip archive	apache-maven-3.9.9-src.zip	apache-maven-3.9.9-src.zip.sha512	apache-maven-3.9.9-src.zip.asc

- 3.9.9 Release Notes and Release Reference Documentation
- latest source code from source repository
- Distributed under the Apache License, version 2.0
- other:
  - All current release sources (plugins, shared libraries,...) available at https://downloads.apache.org/maven/

Maven Wrapper

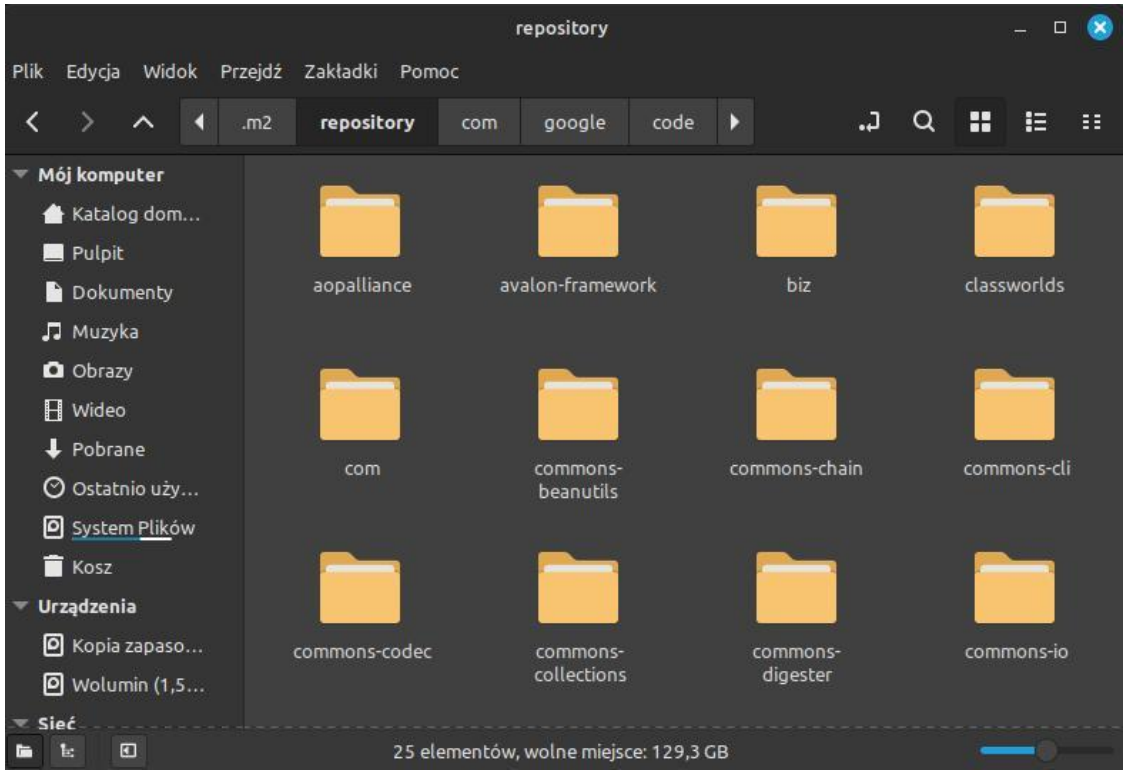
- Maven Wrapper jest skryptem, który można dodać do każdego mavenowego projektu, aby zapewnić jego pełne działanie (jak ze zwyczajnie zainstalowanym Mavenem) bez konieczności posiadania Mavena "globalnie" w systemie. Może on również ułatwić zachowanie spójności jego wersji wśród wszystkich użytkowników projektu.
- Chociaż Maven Wrapper umożliwia posługiwanie się Mavenem bez jego ogólnej instalacji, to jednak jest ona potrzebna do zainicjowania Wrappera w danym projekcie. Projekt z już utworzonymi plikami Maven Wrappera jest gotowy do użycia na wszystkich innych komputerach.

Instalacja i użycie Maven Wrappera

- W pierwszej kolejności należy utworzyć mavenowy projekt według wspomnianych wcześniej instrukcji.
- Później wywołać komendę "mvn wrapper:wrapper".
- Polecenie instalujące tworzy w projekcie dwa pliki: *mvnw* (dla Linuksa i MacOS'a) oraz *mvnw.cmd* (dla Windowsa).
- Od tego momentu można używać Mavena w projekcie w taki sam sposób jak w jego ogólnej wersji, z tą różnicą, że zamiast "mvn" zapisujemy lokalizację pliku wrappera dla odpowiedniego systemu operacyjnego: np. w katalogu głównym projektu: "./mvnw" lub "mvnw.cmd".

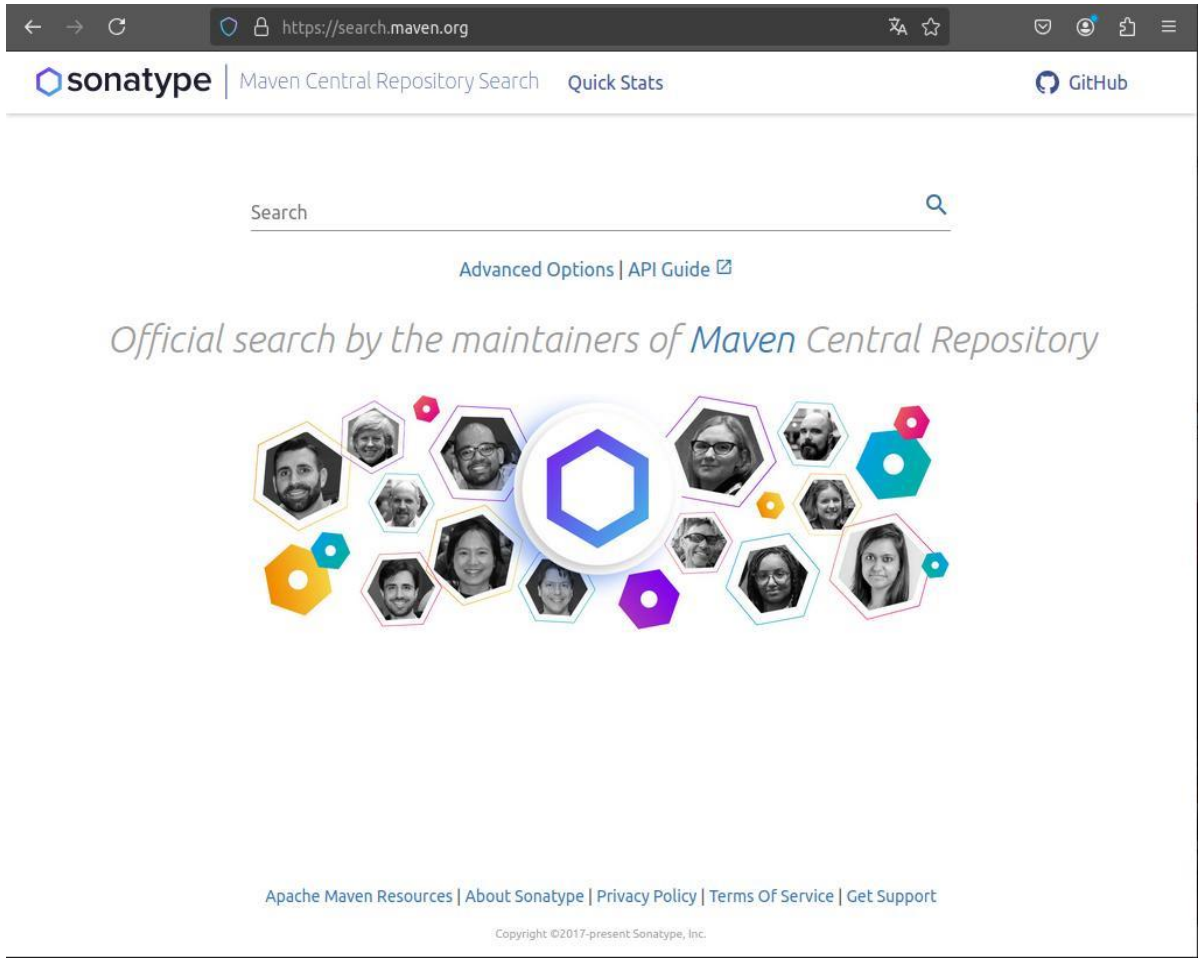
Lokalne repozytoria Mavena (Local Repositories)

- pobrane zależności z repozytoriów zdalnych
- znajdują się w:
  - Windows: C:\Users\<User_Name>\.m2
  - Linux i Mac: ~/.m2
- modyfikować je powinno się poprzez api Mavena – nie ręcznie



Repozytoria zdalne (Remote Repositories)

- sieciowe źródło pobierania artefaktów
- domyślnie Maven Central Repository



Strona wyszukiwania w Maven Central Repository

Snapshot, a Release

Snapshot

- jest wersją pośrednią
- jego metadane zawierają informacje o dacie kompilacji
- Maven sprawdza, czy nie ma w zdalnym repozytorium nowej wersji, z częstotliwością wynikającą z ustawień (domyślnie codziennie)

Release

- jest wersją ostateczną
- próba jego aktualizacji na zdalnym repozytorium (jeżeli się go deploy'uje) powinna zwrócić błąd
- aktualizacja może nastąpić jedynie poprzez wypuszczenie nowej wersji

Generowanie projektu

```
$ mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.5 -
DinteractiveMode=false
```

Archetype - pierwowzór

Polecenie generuje projekt na podstawie szablonu (archetype) oraz podanych argumentów

POM.xml - Project Object Model

- zawiera informacje o projekcie i konfiguracji budowania projektu
- przy uruchomieniu celu (goal) lub zadania (target), Maven szuka go w obecnej lokalizacji

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

	<groupId>com.mycompany.app</groupId>
	<artifactId>my-app</artifactId>
	<version>1</version>
</project>
```

Minimalny POM.xml

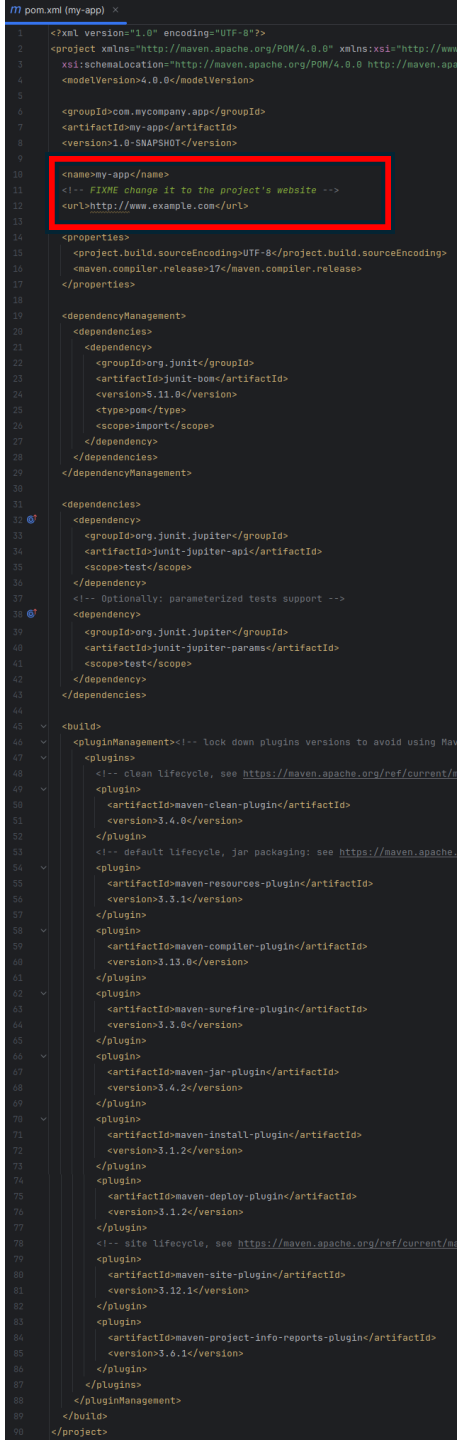
- `<modelVersion>` – wersja modelu POMa (dla obecnie używanych Mavenów 2 i 3 po prostu **4.0.0**)
- `<groupId>` – id grupy projektu
- `<artifactId>` – id projektu
- `<version>` – wersja projektu

POM.xml - Project Object Model

metadane używane np. przy automatycznym generowaniu dokumentacji:

- `<name>` - wyświetlana nazwa projektu
- `<url>` – strona internetowa projektu

```
10      <name>my-app</name>
11      <!-- FIXME change it to the project's website -->
12      <url>http://www.example.com</url>
```



POM.xml - Project Object Model

`<properties>` – dowolnie nazwane zdefiniowane właściwości projektu, których można później używać (zarówno w POM-ie jak i w kodzie Javy). Można też używać odziedziczonych zmiennych.

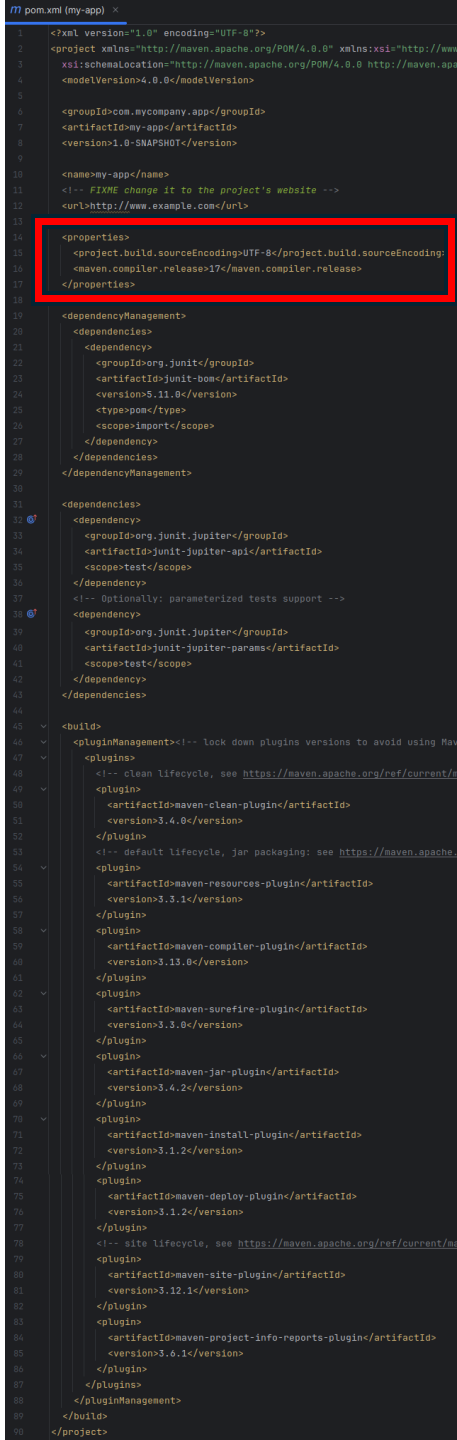
```
14    <properties>
15      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16      <maven.compiler.release>17</maven.compiler.release>
17    </properties>
```

Później można ich używać poprzez składnię ${*nazwa_zmiennej*}:

```
<url>https://www.project-${maven.compiler.release}.agh.edu.pl</url>
```

Ewentualnie można użyć też w kodzie Javy:

https://stackoverflow.com/questions/11500533/access-maven-properties-defined-in-the-pom

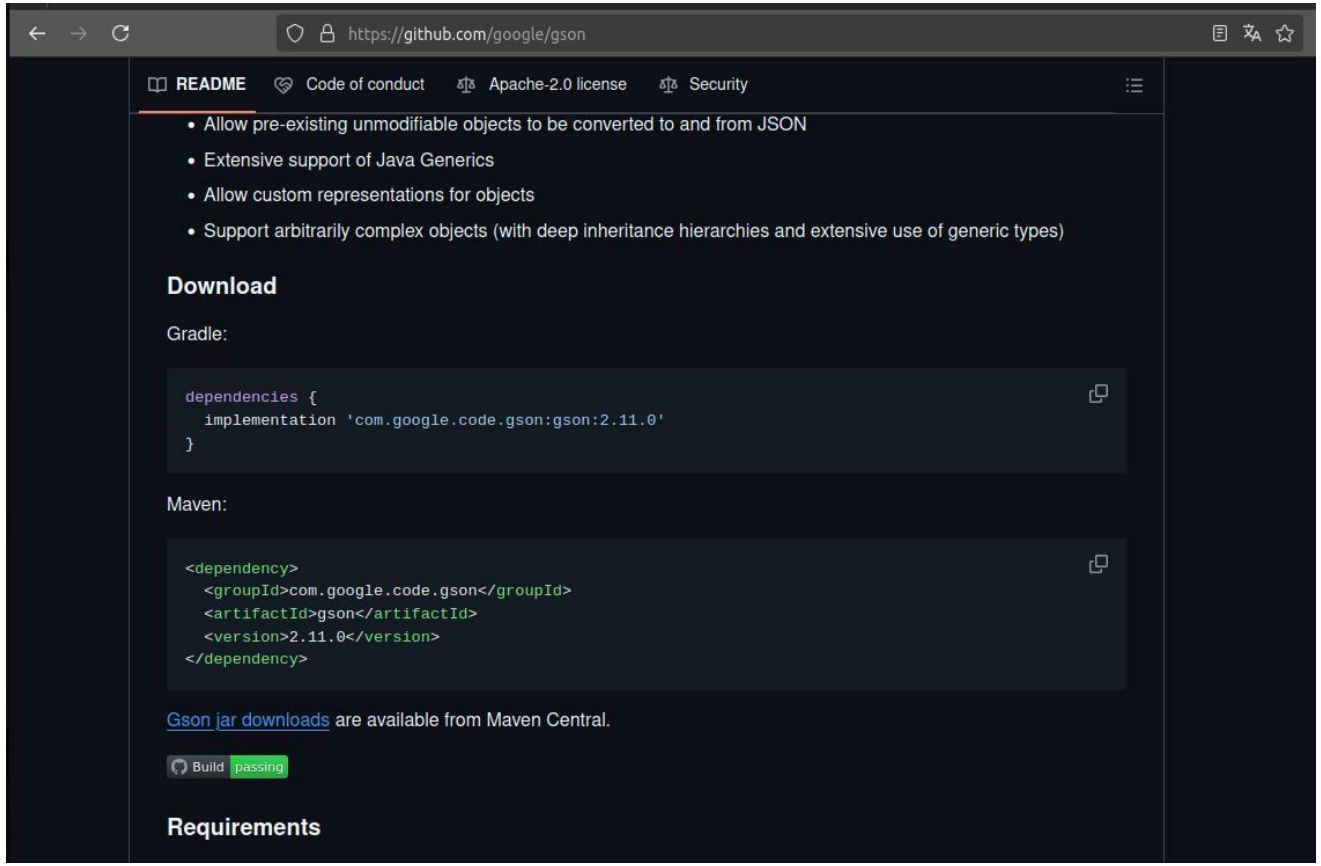


POM.xml - Project Object Model

Zależności projektu - właściwie najczęściej używana sekcja. To tutaj można dodawać, usuwać i zmieniać wersje zależności.

```xml
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <scope>test</scope>
  </dependency>
  <!-- Optionally: parameterized tests support -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Jak dodać zależność?



repozytorium Gsona

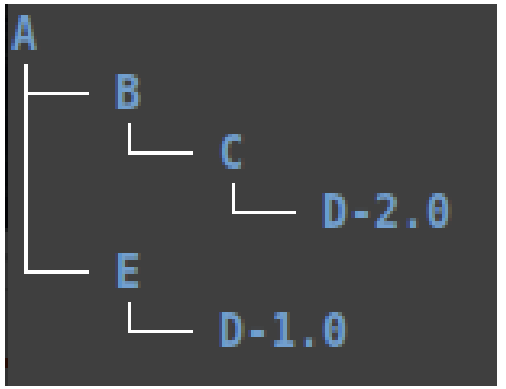
```xml
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <scope>test</scope>
    </dependency>
    <!-- Optionally: parameterized tests support -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-params</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.11.0</version>
    </dependency>
</dependencies>
```

sekcja z zależnościami w POM.xml

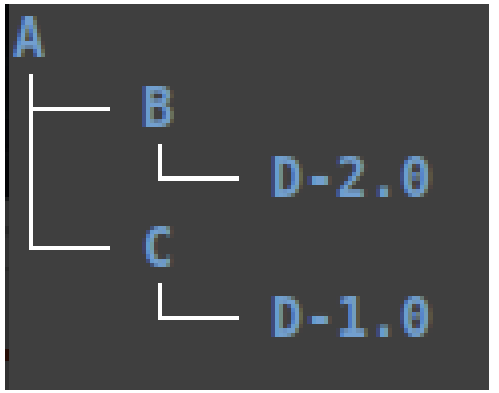
Dziedziczenie zależności

Gdy dwie zależności mają tę samą zależność, ale w innej wersji, to wybierana jest ta, która:

1. jest płycej
2. jest pierwsza w ramach tego samego poziomu



Będzie użyte D w wersji 1.0



Będzie użyte D w wersji 2.0

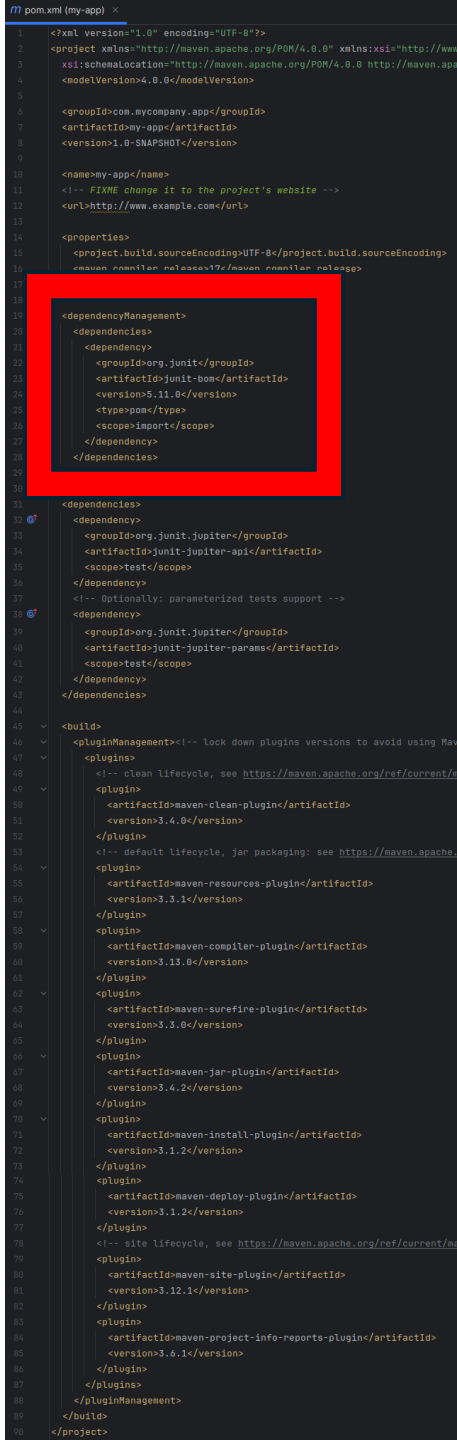
W razie potrzeby można wymusić użycie konkretnej wersji w elemencie `<dependencies>` lub `<dependencyManagement>`

POM.xml - Project Object Model

Sekcja `<dependencyManagement>` różni się od elementu `<dependency>` tym, że taka zależność zostanie dodana do potomnego modułu (child module) tylko wtedy, gdy znajduje się ona w jego sekcji `<dependency>`.

Gdyby zależność umieścić we własnej sekcji `<dependency>`, dodanie zależności nastąpiłoby bez względu na zawartość potomnego POM.xml.

```
19    <dependencyManagement>
20      <dependencies>
21        <dependency>
22          <groupId>org.junit</groupId>
23          <artifactId>junit-bom</artifactId>
24          <version>5.11.0</version>
25          <type>pom</type>
26          <scope>import</scope>
27        </dependency>
28      </dependencies>
29    </dependencyManagement>
```

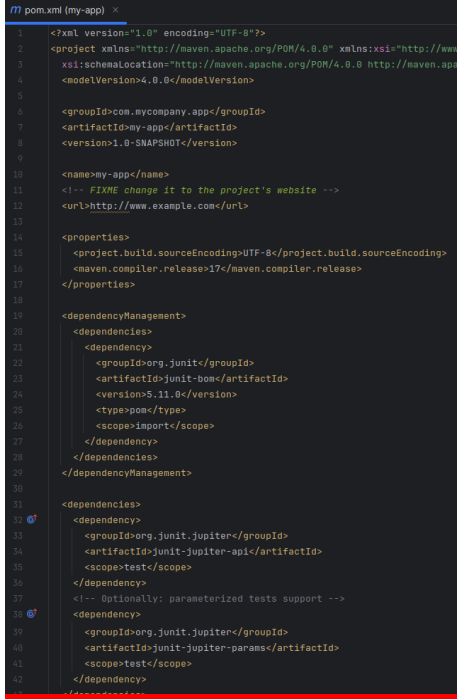


POM.xml - Project Object Model

```
45      <build>
46        <pluginManagement><!-- lock down plugins versions to avoid
47          <plugins>
48            <!-- clean lifecycle, see https://maven.apache.org/ref/
49            <plugin>
50              <artifactId>maven-clean-plugin</artifactId>
51              <version>3.4.0</version>
52            </plugin>
53            <!-- default lifecycle, jar packaging: see https://mave
54            <plugin>
55              <artifactId>maven-resources-plugin</artifactId>
56              <version>3.3.1</version>
57            </plugin>
```

Przykładowe możliwe tagi w `<build>`:

- `<defaultGoal>` - domyślny cel
- `<directory>` - ścieżka docelowa dla zbudowanych plików
- `<finalName>` - nazwa pliku wynikowego (wcale nie taka ostateczna)
- `<plugins>` - konfiguracja używanych pluginów



Pluginy

Wszystko w Mavenie dzieje się za pośrednictwem pluginów. Dlatego każdy cel (goal) wymaga użycia odpowiedniego plugina. Dzielą się na dwa rodzaje:

- build plugins
  - wykonywane podczas budowania projektu
  - konfiguracja jest w elemencie `<build>` w POM-ie
- reporting plugins
  - używane do generowania strony (plugin site) projektu
  - konfiguracja jest w elemencie `<reporting>` zwyczajowo umieszczanym przed elementem `<build>` w POM-ie

clean

- Folder *target*, umieszczony w mavenowym projekcie, zawiera wszystkie pliki będące rezultatem budowania projektu.
- Polecenie "mvn clean" usuwa ten folder. Używa się go, aby uniknąć problemów (np. z czasem wykonania) w przypadku dokonania istotnych zmian, takich jak zmiana nazwy lub usunięcie pliku źródłowego klasy. W takich sytuacjach Maven nie posiada wystarczających informacji do usunięcia danego pliku przy kolejnym budowaniu. W efekcie do końcowego archiwum *jar* mogą trafić "śmieci".
- Polecenia tego do poprawnego działania wymagają również niektóre pluginy.

compile i test

- Polecenie "mvn compile" kompiluje wszystkie klasy znajdujące się w *src\main\java* oraz umieszcza je w odpowiednim pakiecie w podkatalogu *classes* folderu *target*.
- "mvn test" równoznaczne poleceniu "mvn compile test" wywołuje "mvn compile", a następnie uruchamia testy z folderu src*test* projektu, wyświetlając następnie stosowne informacje o rezultacie.

Polecenia te korzystają z konwencji hierarchii plików projektów mavenowych (więcej: maven.apache.org/...).

package

"mvn package" z kolei opiera się na poprzednich dwóch poleceniach. Daje wynik identyczny jak "mvn compile test package", a więc uruchamia kompilację i testy, a następnie pakuje cały kod w archiwum z rozszerzeniem *.jar* i umieszcza go we wspomnianym wcześniej *target*.

Aby poprzednie pliki kompilacji nie zostały ponownie wykorzystane, możliwe jest użycie "mvn clean compile test package"/"mvn clean package".

install

"mvn install" wykonuje *czystą instalację* projektu - usuwa folder *target* (polecenie "mvn clean"), ponownie buduje projekt od początku, uruchamia testy i tworzy archiwum .*jar* (polecenie "mvn package"), a następnie umieszcza go w lokalnym repozytorium Mavena obok wszystkich zależności.

Więcej szczegółów o pluginach: maven.apache.org/plugins

W tym "site" do tworzenia dokumentacji: site plugin

Mavenowy cykl życia (lifecycle)

Proces budowania i dystrybucji projektu (artefaktu) tworzonego z pomocą Mavena jest ściśle określony. Dzięki temu konieczna jest znajomość stosunkowo niewielu poleceń. Zamierzony rezultat pomaga osiągnąć POM.

Trzy wbudowane cykle życia:

- default (domyślny, wdrażanie projektu)
- clean (czyszczenie projektu)
- site (tworzenie strony projektu)

Clean Lifecycle

Phase	Description
`pre-clean`	execute processes needed prior to the actual project cleaning
`clean`	remove all files generated by the previous build
`post-clean`	execute processes needed to finalize the project cleaning

Site Lifecycle

Phase	Description
`pre-site`	execute processes needed prior to the actual project site generation
`site`	generate the project's site documentation
`post-site`	execute processes needed to finalize the site generation, and to prepare for site deployment
`site-deploy`	deploy the generated site documentation to the specified web server

maven.apache.org

Default Lifecycle 1

Phase	Description
`validate`	validate the project is correct and all necessary information is available.
`initialize`	initialize build state, e.g. set properties or create directories.
`generate-sources`	generate any source code for inclusion in compilation.
`process-sources`	process the source code, for example to filter any values.
`generate-resources`	generate resources for inclusion in the package.
`process-resources`	copy and process the resources into the destination directory, ready for packaging.
`compile`	compile the source code of the project.
`process-classes`	post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.

Default Lifecycle 2

`generate-test-sources`	generate any test source code for inclusion in compilation.
`process-test-sources`	process the test source code, for example to filter any values.
`generate-test-resources`	create resources for testing.
`process-test-resources`	copy and process the resources into the test destination directory.
`test-compile`	compile the test source code into the test destination directory
`process-test-classes`	post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes.
`test`	run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.

Default Lifecycle 3

`prepare-package`	perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package.
`package`	take the compiled code and package it in its distributable format, such as a JAR.
`pre-integration-test`	perform actions required before integration tests are executed. This may involve things such as setting up the required environment.
`integration-test`	process and deploy the package if necessary into an environment where integration tests can be run.
`post-integration-test`	perform actions required after integration tests have been executed. This may including cleaning up the environment.
`verify`	run any checks to verify the package is valid and meets quality criteria.
`install`	install the package into the local repository, for use as a dependency in other projects locally.
`deploy`	done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Przyspieszanie budowania

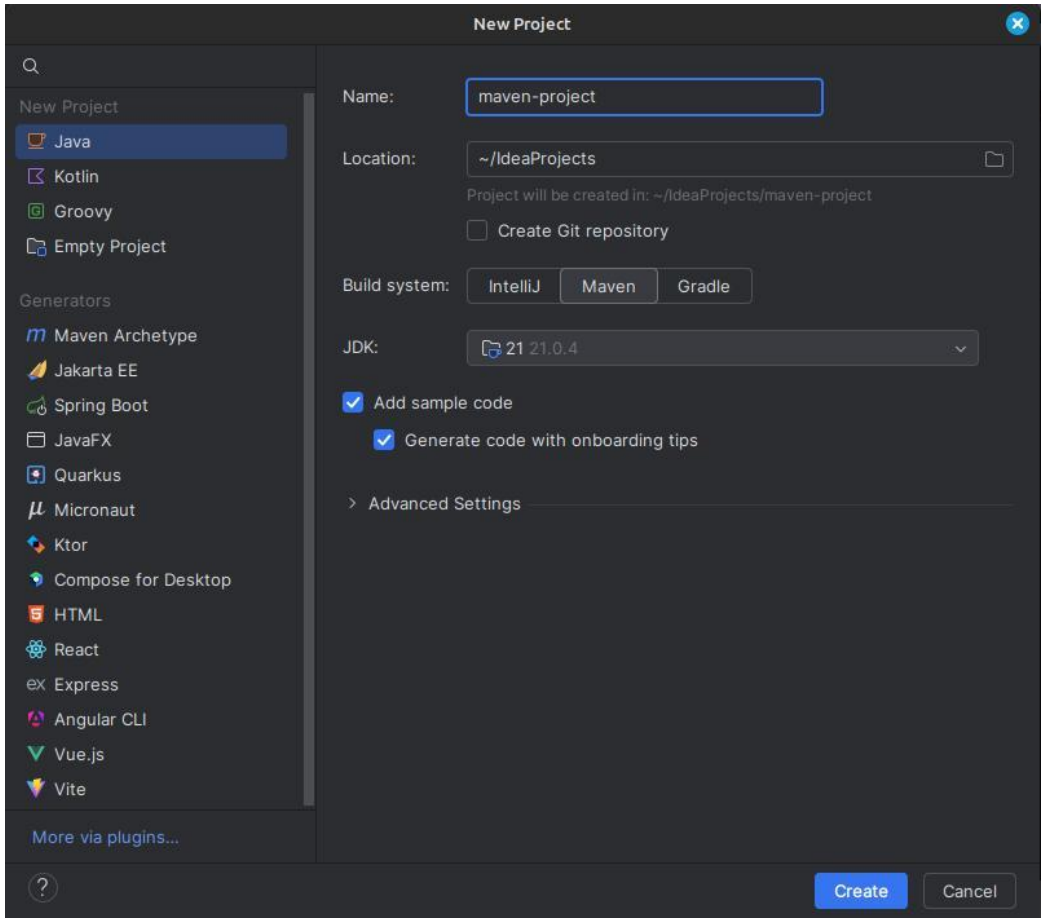
- **uruchomienie budowania na wielu wątkach**: "mvn clean install -T", gdzie po spacji podajemy liczbę wątków, bazując na możliwościach naszej maszyny ('C' po liczbie oznacza liczbę wątków na rdzeń procesora)

Pominięcie testów:

- "mvn clean install -DskipTests"
- "mvn clean install -Dmaven.test.skip -DskipTests"
- "mvn -T 1C clean install -Dmaven.test.skip -DskipTests"
- **utrzymywanie Javy w najnowszej wersji**
- **Dostosowanie pamięciowej konfiguracji w pliku *mvn.bat*:** MAVEN_OPTS=-Xmx512m -XX:MaxPermSize=256m

Jak to często bywa, kombinować można do woli: inne sposoby (stackoverflow)

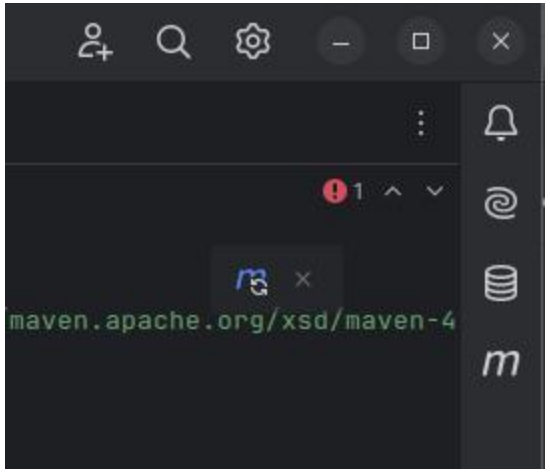
Nowy projekt z Mavenem w IntelliJ'u



zakładka Maven w Build System

Brak pobranej zależności w lokalnym repozytorium

błąd pokazujący się w IntelliJu



Należy nacisnąć ikonkę przeładowania Mavena – wtedy błędy znikają.

Źródła:

- https://maven.apache.org/
- https://maven.apache.org/what-is-maven.html
- https://stackoverflow.com/questions/5112607/how-to-include-libraries-in-java-without-using-an-ide
- https://github.com/google/gson
- https://mvnrepository.com/artifact/com.google.code.gson/gson
- https://home.agh.edu.pl/~skrzynia/pz1/lab1.pdf
- https://google.github.io/gson/UserGuide.html
- https://github.com/google/gson
- https://search.maven.org/artifact/com.google.code.gson/gson/2.11.0/jar?eh=
- https://maven.apache.org/repositories/local.html
- https://www.baeldung.com/maven-local-repository
- https://www.baeldung.com/maven-snapshot-release-repository
- https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
- https://maven.apache.org/guides/getting-started/index.html
- https://maven.apache.org/guides/introduction/introduction-to-the-pom.html
- https://maven.apache.org/pom.html
- https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html
- https://maven.apache.org/plugins/index.html
- https://maven.apache.org/ref/3.9.9/maven-core/lifecycles.html#clean_Lifecycle
- https://www.geeksforgeeks.org/a-quick-guide-to-maven-wrapper/
- https://maven.apache.org/wrapper/
- https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html
- https://www.youtube.com/watch?app=desktop&v=Xatr8AZLOsE
- https://stackoverflow.com/questions/4662452/in-maven-why-run-mvn-clean