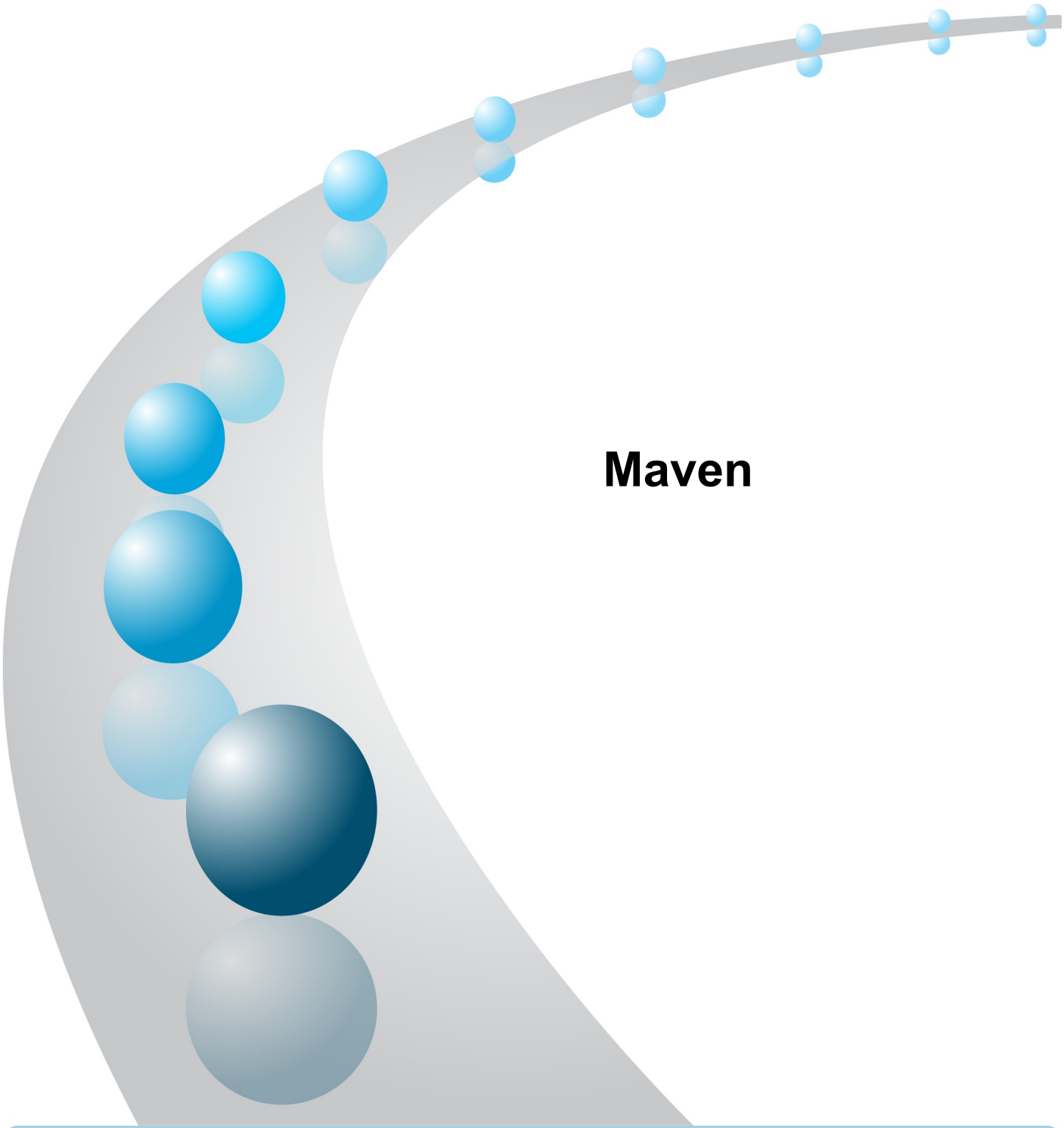
Noël Vaes
Java Trainer & Consultant



Maven

Roode Roosstraat 5
3500 Hasselt
België

+32 474 38 23 94
noel@noelvaes.eu
www.noelvaes.eu

Noël Vaes
Roode Roosstraat 5
3500 Hasselt
België

Tel: +32 474 38 23 94

noel@noelvaes.eu
www.noelvaes.eu

08/01/2019

Inhoudsopgave

Hoofdstuk 1: Maven

1.1. Inleiding

Maven is een projectmanagement-*tool*. Het is een *tool* waarmee men een Java-project op een gestandaardiseerde wijze kan vormgeven en beheren. Dit behelst onder andere het *builden* (compileren, JAR maken enzovoort) van een project maar ook het genereren van rapporten en documentatie, het maken van een website enzovoort.
Maven is dus meer dan een *build tool* zoals *ANT*, maar alle mogelijkheden van een *build tool* zijn wel voorzien.

In de volgende paragrafen zullen we stap voor stap de mogelijkheden van *Maven* verkennen aan de hand van praktische voorbeelden.

We zullen *Maven* hier behandelen in combinatie met JDK 11.

1.2. *Maven* installeren en configureren

Maven is een *open-source*-project van *Apache* en is te vinden op volgende website: *http://maven.apache.org*. *Maven k*an geïnstalleerd worden door het bestand **apache-maven-3.x.y-bin.zip** af te halen en uit te pakken op het lokale systeem.

Opdracht 1: *Maven* installeren

- Haal het bestand **apache-maven-3.x.y-bin.zip** van de website http://maven.apache.org.
- Pak dit bestand uit op je lokale systeem, bijvoorbeeld in **C:\Program Files**
- Voeg de volgende omgevingsvariabele toe aan het besturingssysteem:

  ```
  MAVEN_HOME="C:\Program Files\apache-maven-3.x.y"
  ```

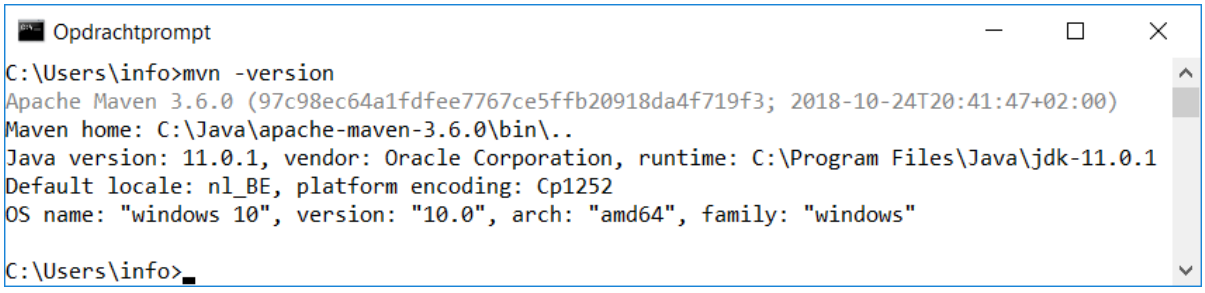
- Voeg tevens de plaats van *Maven* toe aan de variabele `PATH` zodat we *Maven* kunnen uitvoeren vanop de commandolijn:

  ```
  PATH=...;%MAVEN_HOME%\bin
  ```

- Zorg er tevens voor dat de omgevingsvariabele `JAVA_HOME` verwijst naar de installatie van de JDK (en niet naar de JRE).

- Open een commandovenster en voer het volgende commando uit:

  ```
  mvn -version
  ```



1.3. Mijn eerste *Maven*-project

Na een succesvolle installatie kunnen we *Maven* beginnen te gebruiken voor het beheer van Java-projecten.

Een project wordt beschreven in een *Project Object Model*. Dit is een XML-bestand met de naam ***pom.xml*** dat zich in de hoofdmap van het project dient te bevinden.

Java-projecten bevatten doorgaans afzonderlijke mappen voor de broncode, bibliotheken (JAR-bestanden), gecompileerde klassen enzovoort. *Maven* maakt het beheer van projecten eenvoudiger door deze mappenstructuur te standaardiseren. Het is mogelijk, maar niet aangewezen om hiervan af te wijken. Het gebruik van de *Maven*-standaardinstellingen heeft als voordeel dat de configuratie minimaal is en dat men makkelijk inzicht krijgt in nieuwe projecten die ook deze standaard volgen.

Voor een eenvoudig Java-project ziet deze mappenstructuur er als volgt uit:

```
project
  +--src
      +--main
          +--java
          +--resources
      +--test
          +--java
          +--resources
  +-- target
        +--classes
        +--test-classes
  pom.xml
```

Map/bestand	***Inhoud***
`src/main/java`	De Java-broncode van het project.
`src/main/resources`	Andere bestanden die we nodig hebben, zoals *property*-bestanden, configuratiebestanden, afbeeldingen enzovoort.
`src/test/java`	De Java-broncode van de testklassen (*JUnit* of *TestNG*)
`src/test/resources`	Andere bestanden die we enkel nodig hebben in de testklassen.
`target`	Gegenereerde *artifacts*: JAR - WAR - EAR - ZIP.
`target/classes`	De gecompileerde klassen.
`target/test-classes`	De gecompileerde testklassen.
`pom.xml`	Dit bestand bevat het *Project Object Model*.

Terwijl Java-broncodebestanden uit de map ***src/main/java*** gecompileerd worden naar de map ***target/classes*** worden de *resource*-bestanden uit de map ***src/main/resource*** gekopieerd naar de map ***target/classes***. Dit impliceert dat deze bestanden in het finale *classpath* beschikbaar worden gesteld.

We zullen dit alles illustreren met een voorbeeld. Stel dat we een project willen maken voor de alom bekende "Hello World". We maken hiervoor de volgende mappenstructuur:

```
project
   +--src
      +--main
         +--java
            +--module-info.java
            +--eu
               +--noelvaes
                  +--hello
                     +--App.java
   pom.xml
```

De broncode:

App.java

```
package eu.noelvaes.hello;

/**My own application class.
 *
 * @author No&euml;l Vaes
 *
 */
public class App {
   /**This method says hello to the world.
    * @return "Hello World"
    */
   public String sayHello() {
      return "Hello World";
   }

   public static void main(String[] args) {
      App app = new App();
      System.out.println(app.sayHello());
   }
}
```

De modulebeschrijving:

module-info.java

```
module eu.noelvaes.hello {
}
```

Onze module krijgt hier de naam ***eu.noelvaes.hello***. We exporteren geen pakketten aangezien het hier om een hoofdprogramma gaat.

Het POM-bestand:

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
   xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <modelVersion>4.0.0</modelVersion>
   <groupId>eu.noelvaes.hello</groupId>
```

```
   <artifactId>Hello</artifactId>
   <version>1.0</version>

   <name>Hello</name>

   <properties>
      <project.build.sourceEncoding>
         UTF-8
      </project.build.sourceEncoding>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
   </properties>

   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.0</version>
         </plugin>
       </plugins>
    </build>
</project>
```

Het compileren en het inpakken in een JAR-bestand kan met het volgende commando gebeuren:

`mvn package`

Bij de uitvoering van dit commando wordt de broncode gecompileerd en ingepakt in het volgende JAR-bestand: ***Hello-1.0.jar*** in de map ***target***.

Er is weinig configuratie nodig omdat we hier gebruikmaken van de *Maven*-projectstructuur. *Maven* zoekt de broncode in de map ***src/main/java***, compileert de klassen naar de map ***target/classes*** en maakt een JAR-bestand in de map ***target***.

Om de gecompileerde klassen en het JAR-bestand te verwijderen en weer van een schone lei te beginnen, gebruiken we het volgende commando:

`mvn clean`

Men kan de commando's ook na elkaar plaatsen zodat ze in de aangegeven volgorde worden uitgevoerd. Heel gebruikelijk is eerst alles wissen en vervolgens alles opnieuw bouwen:

```
mvn clean package
```

Merk op dat bij het eerste gebruik van *Maven* allerlei modules van het internet gehaald worden. Deze worden opgeslagen in de lokale *repository* zodat ze voortaan onmiddellijk beschikbaar zijn. Hierover later meer.

Opdracht 2: Mijn eerste project

In deze opdracht gaan we ons eerste *Maven*-project maken. We kunnen de benodigde mappenstructuur handmatig aanmaken maar we kunnen dat ook doen via een ingebouwd *archetype*.

- Maak een map met de naam ***CursusMaven*** met hierin een supmap ***Hello***.
- Maak in de map ***Hello*** de nodige submappen met het volgende commando:

```
mkdir src\main\java\eu\noelvaes\hello
```

- Voeg het bestand ***src\main\java\eu\noelvaes\hello\App.java*** toe met de volgende inhoud :

```
package eu.noelvaes.hello;

/**My own application class.
 *
 * @author No&euml;l Vaes
 *
 */
public class App {
   /**This method says hello to the world.
    * @return "Hello World"
    */
   public String sayHello() {
      return "Hello World";
   }

   public static void main(String[] args) {
      App app = new App();
      System.out.println(app.sayHello());
   }
}
```

- Voeg het bestand ***src\main\java\module-info.java*** toe met de volgende inhoud:

```
module eu.noelvaes.hello {
}
```

- Voeg ten slotte het POM-bestand ***pom.xml*** toe in de projectmap:

```
<?xml version="1.0" encoding="UTF-8"?>
<project
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
```

```
   xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <modelVersion>4.0.0</modelVersion>
   <groupId>eu.noelvaes.hello</groupId>
   <artifactId>Hello</artifactId>
   <version>1.0</version>

   <name>Hello</name>

   <properties>
      <project.build.sourceEncoding>
         UTF-8
      </project.build.sourceEncoding>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
   </properties>

   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.0</version>
         </plugin>
       </plugins>
    </build>

</project>
```

- Open een commandovenster op de plaats van het POM-bestand en voer het volgende commando uit:

  `mvn package`

- Ga na of in de map ***target*** de compileerde klassen en het JAR-bestand aanwezig zijn.
- Voer vervolgens het volgende commando uit:

  `mvn clean`

- Ga na of de map ***target*** leeggemaakt is. Bouw het project opnieuw met **`mvn package`**.

1.4. Project Object Model

Het *Project Object Model* wordt beschreven in het bestand ***pom.xml*** en ziet er voor een eenvoudig project als volgt uit:

```
<?xml version="1.0" encoding="UTF-8"?>
<project
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
   xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <modelVersion>4.0.0</modelVersion>
   <groupId>groupId</groupId>
   <artifactId>artifactId</artifactId>
   <version>x.y.z</version>
```

```
</project>
```

In de onderstaande tabel overlopen we de verschillende verplichte *tags*:

Tag	***Beschrijving***
`<modelVersion>`	Geeft het versienummer van de POM weer. Momenteel is dat steeds 4.0.0 maar dat zou in de toekomst kunnen wijzigen bij nieuwe versies van *Maven*.
`<groupId>`	Identificeert de eigenaar van dit project. Het is gebruikelijk hiervoor de domeinnaam te gebruiken, net zoals dat bij pakketdefinities gebeurt.
`<artifactId>`	Identificeert dit project.
`<version>`	De versie van dit project.

De *tags* `<groupId> <artifactId>` en `<version>` vormen de coördinaten van het project. Zij identificeren het project op unieke wijze.

Ieder project heeft een versienummer dat onder andere gebruikt wordt door andere projecten om aan te geven welke versie van een module nodig is.

Het versienummer bestaat uit vier delen:

`major.minor.incremental-qualifier`

`major`	Het hoofdnummer van het project.
`minor`	Het nevennummer van het project.
`incremental`	Het incrementele nummer van het project.
`qualifier`	Een extra kwalificatie van het project.

Voorbeelden

1.0
1.2.3
2.0.6-alpha
2.0.3-beta

Versies worden geordend volgens dit nummeringssysteem. De *qualifiers* worden hierbij alfabetisch geordend.

Het nummer kan de tekst **SNAPSHOT** bevatten en hiermee wordt aangegeven dat het project nog in ontwikkeling is. De tekst "SNAPSHOT" wordt door *Maven* bij het installeren van het *artifact* vervangen door de datum en tijd. Indien een ander project afhankelijk is van zo'n *snapshot*-module zal er regelmatig nagegaan worden of er een nieuwe versie beschikbaar is.

In ons voorbeeld hebben we nog extra configuratie toegevoegd die te maken heeft met het gebruik van *Maven* in combinatie met JDK 11. We zullen de details hiervan later behandelen. Op dit moment hebben we deze configuratie reeds nodig.

1.5. Dependencies

Een project kan afhankelijk zijn van een ander project. Het is immers gebruikelijk zo veel mogelijk beroep te doen op reeds bestaande code. Zeker in *open-source*-projecten wordt er