



samen sterk voor werk

Maven



Deze cursus is eigendom van de VDAB©

Inhoudsopgave

1	INLEIDING.....	2
1.1	Doelstelling.....	2
1.2	Vereiste voorkennis.....	2
1.3	Nodige software	2
2	PROJECT STRUCTUUR.....	3
2.1	pom.xml.....	4
2.2	Een eerste Maven project	4
3	DEPENDENCIES	6
3.1	De dependency zoeken in de Maven Central Repository	6
3.2	De dependency toevoegen aan je project	6
3.3	Version range	9
3.4	Scope	9
4	LIFE CYCLE.....	11
4.1	clean	11
4.2	default	11
4.3	site	12
5	COLOFON.....	15

1 INLEIDING

1.1 Doelstelling

Je leert werken met Maven: een standaard voor Java projecten.

Maven lost twee problemen op:

- ➖ Elke Java editor (NetBeans, Eclipse, IntelliJ, ...) heeft een eigen directory structuur voor een Java project en heeft ook eigen configuratiebestanden voor een Java Project. Dit maakt het moeilijk om een project dat je met één van deze editors gemaakt hebt te openen in een andere editor. Het is ook moeilijk toe te laten dat de leden van een softwareteam verschillende editors gebruiken om aan een gemeenschappelijk project te werken. Je leert in deze cursus Maven kennen in samenwerking met NetBeans. Je leert in de volgende cursussen Maven kennen in samenwerking met Eclipse.
- ➖ Er bestaan duizenden open source libraries voor Java. Het is niet eenvoudig deze libraries te zoeken en te downloaden omdat ze zich bevinden op verschillende websites.

Maven biedt een oplossing:

- ➕ Een Maven project heeft een standaard directory structuur en een standaard configuratiebestand. Gezien de populariteit van Maven kan je een Maven openen met elke Java editor. Deze editors kennen de directory structuur en het configuratiebestand van Maven en kunnen er perfect mee samenwerken.
- ➕ De meeste populaire Java libraries zijn verzameld op de centrale website van Maven (<https://maven.apache.org>). Als je een library nodig hebt, vind je hem hier. Deze website wordt ook de "Maven central repository" genoemd.



Dit is een introductiecursus over Maven. Je zal in de volgende cursussen van het traject dieper ingaan op bepaalde aspecten van Maven.

1.2 Vereiste voorkennis

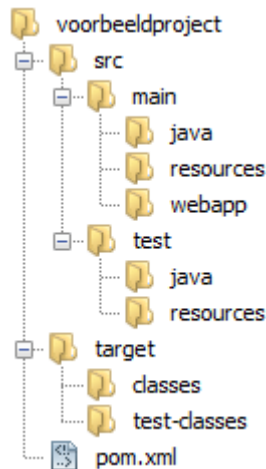
- Java
- JDBC

1.3 Nodige software

- een JDK (Java Developer Kit) met versie 8 of hoger.
- NetBeans

2 PROJECT STRUCTUUR

De standaard project structuur van Maven is als volgt:



- **src/main/java**
Hier komen de Java sources.
- **src/main/resources**
Hier komen bestanden die je gebruikt in het project, maar geen Java sources zijn.
Voorbeelden: afbeeldingen, XML configuratiebestanden, ...
- **src/main/webapp**
Hier komen bestanden als je project een website is.
Voorbeelden: HTML bestanden, CSS bestanden, JavaScript bestanden.
Je leert hier alles over in de cursus Spring fundamentals.
- **src/test/java**
Hier komen Java sources met geautomatiseerde testen.
Je leert hier alles over in de cursus JUnit.
- **src/test/resources**
Hier komen bestanden die nodig zijn voor de testen, maar geen Java sources zijn.
Voorbeelden: XML configuratiebestanden, ...
- **target**
Hier komt het eindproduct van het project.
Dit kan een JAR (Java Archive) bestand zijn. Dit is een gecomprimeerd bestand dat alle gecompileerde bestanden van je project bevat en ook andere bestanden die je project nodig heeft (zoals afbeeldingen). Als het project een website is, zal het een WAR (Web Application Archive) bestand zijn. Dit is een gecomprimeerd bestand dat alle gecompileerde bestanden van je project bevat en ook andere bestanden die je website nodig heeft: HTML bestanden, CSS bestanden, JavaScript bestanden, afbeeldingen, ...
- **target/classes**
Hier komen de gecompileerde bestanden.
- **target/test-classes**
Hier komen de gecompileerde bestanden van de geautomatiseerde testen.
- **pom.xml**
Dit is het configuratiebestand van je project. De naam van dit bestand ligt vast.
Dit bestand bevat ook een opsomming van de libraries die je in je project gebruikt.

Het is niet noodzakelijk dat je project alle bovenvermelde mappen bevat. Als je project bijvoorbeeld geen geautomatiseerde testen heeft, ontbreekt de map `src/test` en zijn submappen.

2.1 pom.xml

Dit is het configuratiebestand van je project. Pom is een afkorting voor Project Object Model.

Dit bestand kan volgende inhoud hebben:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>be.vdab</groupId>
  <artifactId>hallowereid</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
</project>
```

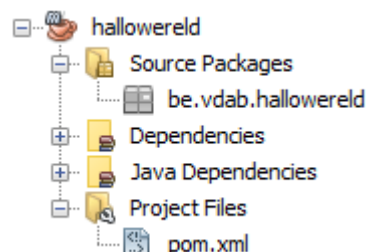
- (1) Hier staat het versienummer van de POM. Bij de huidige versie van Maven is dit 4.0.0.
- (2) Hier staat de eigenaar van het project. Je duidt de eigenaar aan met zijn domeinnaam.
- (3) Hier staat de naam van het project.
- (4) Hier staat het versienummer van het project.
Een versienummer heeft volgende structuur: major.minor.revision-qualifier.
Het major getal wijzigt als het project grondig gewijzigd of uitgebreid is sedert de vorige versie.
Het minor getal wijzigt als het project lichtjes gewijzigd of uitgebreid is sedert de vorige versie.
Het revision getal wijzigt als je fouten gecorrigeerd hebt sedert de vorige versie.
Qualifier is een woord (alfa, beta, ...). Dit wijst er op dat het een proefversie van het project is.
In het voorbeeld is de major 1, de minor 0 en de revision 0 en is er geen qualifier.
- (5) Hier staat in welk soort bestand het eindproduct van het project verpakt wordt.
- (6) Hier staat met welke codering de sources zijn uitgedrukt. De codering UTF-8 kan alle menselijke tekens ter wereld voorstellen en is daarom de meest gebruikte codering.
- (7) en (8) Standaard gaat Maven er van uit dat je Java 5 gebruikt.
Je geeft hier aan dat je Java 8 gebruikt.

2.2 Een eerste Maven project

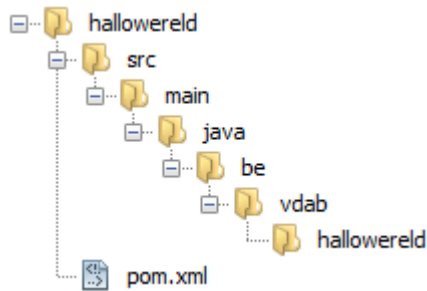
Je maakt in NetBeans een eerste Maven project:

1. Je kiest in het menu File de opdracht New Project.
2. Je kiest Maven bij Categories.
3. Je kiest Java Application bij Projects.
4. Je kiest Next.
5. Je tikt hallowereid bij Project Name. Deze informatie komt in pom.xml (artifactId)
6. Je tikt be.vdab bij Group Id. Deze informatie komt in pom.xml. (groupId)
7. Je tikt 1.0.0 bij Version. Deze informatie komt in pom.xml. (version)
8. Je kiest Finish.

Je project ziet er als volgt uit in het tabblad Projects (links in NetBeans):



Je ziet in het tabblad Files (naast Projects) de mappen waaruit het project echt bestaat:



Je kan pom.xml dubbelklikken om de inhoud te zien.

Je voegt een Main class toe aan het project:

1. Je kiest links het tabblad Projects.
2. Je klikt met de rechtermuisknop op halloweeneld.
3. Je kiest New, Other.
4. Je kiest Java bij Categories.
5. Je kiest Java Main Class bij File Types.
6. Je kiest Next.
7. Je tikt Main bij Class Name.
8. Je kiest be.vdab.halloweeneld bij Package.
9. Je kiest Finish.

Je tikt volgende opdracht in de method main:

```
System.out.println("hallo");
```

Je klikt met de rechtermuisknop op Main in het tabblad Projects en je kiest Run File.

Je ziet in het venster Output (onder in NetBeans) dat het project "gebuid" (gebouwd) wordt.

Hierbij wordt de source file gecompileerd. Daarna voert NetBeans het programma uit en toont de output van het programma ook in hetzelfde venster Output.

3 DEPENDENCIES

Een dependency is een Java library die je gebruikt in je applicatie.

Op dit moment gebruik je nog geen libraries. Je ziet dit als je in het tabblad Projects klikt op het plus teken voor Dependencies: er gebeurt niets.


Je hebt natuurlijk altijd de standaard Java libraries ter beschikking.

Je ziet dit als je in het tabblad Projects klikt op het plus teken voor Java Dependencies.

Je zal nu een eerste dependency toevoegen aan je project. Deze dependency heet commons-text en bevat een heleboel functionaliteit om Strings te manipuleren. Dat deze dependency bestaat en de naam commons-text heeft leer je van een collega, in een boek, op het internet,

3.1 De dependency zoeken in de Maven Central Repository

Je zoekt deze dependency in de Maven Central Repository:

1. Je surft naar de zoekpagina van de Maven Central Repository: <https://search.maven.org>.
2. Je tikt commons-text in het tekstvak en je drukt Enter.
3. Je ziet een lijst met de dependencies die in hun artifactId de tekst commons-text bevatten. Jij hebt de dependency met de GroupId org.apache.commons nodig. Je leert dit terug van een collega, in een boek, op het internet, ...
4. Je ziet het nummer van de laatste versie van deze dependency in de kolom Latest Version. Je ziet daarnaast tussen ronde haakjes het aantal versies (dus inclusief oudere versies) van de dependency.
5. Je klikt het nummer met de laatste versie aan. Je komt op een detailpagina van de dependency. Je ziet een XML fragment onder de tekst Apache Maven (rechts). Dit fragment begint met <dependency> en eindigt met </dependency>. Dit fragment is de identificatie van de library en bevat een groupId, artifactId en version.
6. Je kopieert dit fragment op het klembord met .

3.2 De dependency toevoegen aan je project

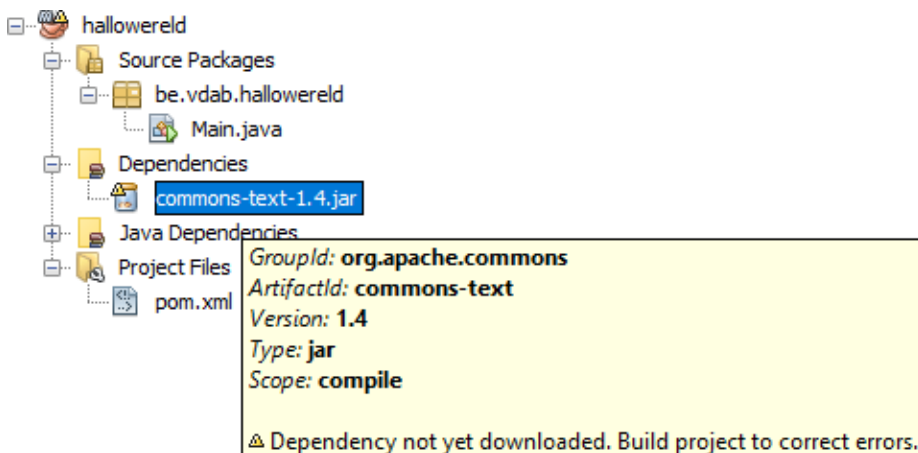
Je voegt deze dependency toe aan je project:

1. Je opent in NetBeans pom.xml.
2. Je tikt volgende regels voor </project>. Deze regels omsluiten al je dependencies.
<dependencies>
</dependencies>
3. Je maakt een blanco regel tussen <dependencies> en </dependencies> en je plakt de inhoud van het klembord.
4. Je klikt met de rechtermuisknop in de XML en je kiest Format om de XML op te maken.
5. Je slaat het bestand op.

Je dubbelklikt in het tabblad Projects het plus teken voor Dependencies.

Je ziet het JAR bestand (library) die de toegevoegde dependency voorstelt.

Als je de muisaanwijzer laat rusten op dit JAR bestand zie je detailinformatie:

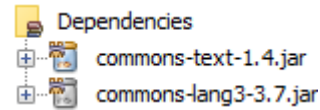




Opmerking: op het moment dat deze cursus werd aangemaakt was 1.4 de laatste versie van de library. Het is mogelijk dat jij een recentere versie hebt.

Je geeft in het tabblad Projects een rechtermuisklik op hallowereid en je kiest Build. NetBeans downloadt nu de dependency die je toevoegde aan pom.xml.

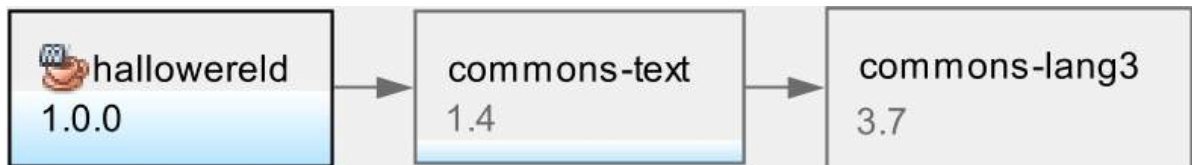
Je ziet in het tabblad Projects in het onderdeel Dependencies dat er 2 libraries zijn toegevoegd aan je project.



- **commons-text-1.4.jar**
Dit is de library die hoort bij de dependency die je toevoegde aan pom.xml.
- **commons-lang3-3.7.jar**
Deze library wordt intern gebruikt door de library commons-text en is automatisch ook gedownload van de Maven Central Repository.
Zo'n library die opgeroepen wordt door een andere library heet een transitive dependency.

Je kan de afhankelijkheid van jouw applicatie op de library commons-text en zijn afhankelijkheid op de library commons-lang op een grafische manier zien:

1. Je klikt boven het venster waar de XML van pom.xml openstaat op Graph.
2. Je klikt daarnaast op Show Graph.
3. Je ziet (mits wat schuiven met de rechthoeken) het volgende:



De pijlen wijzen op een afhankelijkheid (dependency).

Als je in het tabblad Projects op het plus teken klikt voor commons-text-1.2.jar zie je inhoud van de library. Vooral de namen van de packages (die beginnen met org.apache) zijn interessant.

Als je op het plus teken voor een package klikt, zie je de interfaces en classes in die package.

Je gebruikt nu een functionaliteit uit de commons-text library in je programma.

Je wijzigt de class Main:

```

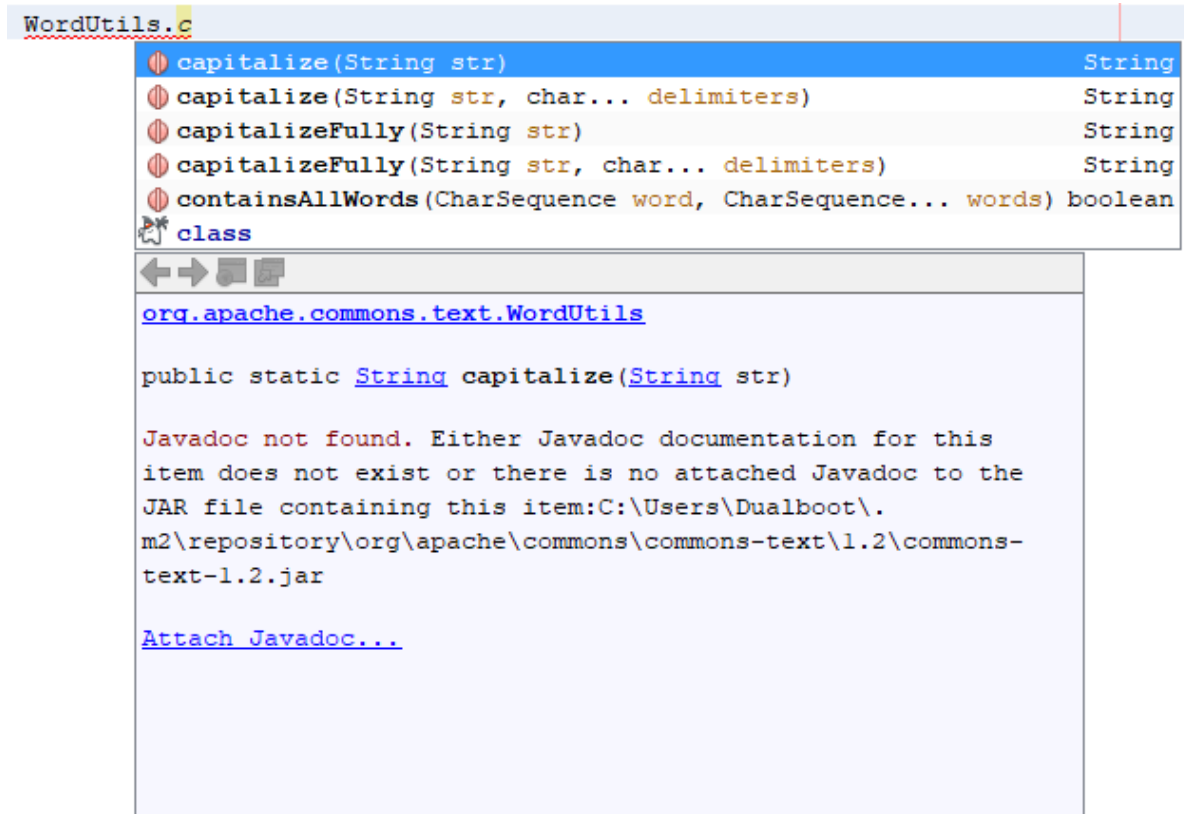
package be.vdab.hallowereid;
import org.apache.commons.text.WordUtils;
public class Main {
    public static void main(String[] args) {
        System.out.println(WordUtils.capitalize("hallo"));
    }
}
  
```

①
②

- (1) Je importeert de class WordUtils uit de package org.apache.commons.text.
- (2) Je roept de static method capitalize op en je geeft een String mee.
De method geeft een String terug die gelijk is aan de meegegeven String, maar waarbij de eerste letter in hoofdletter staat.

Je voert het programma opnieuw uit.

Terwijl je code tikt, krijg je nog geen uitleg over de werking van classes en methods:

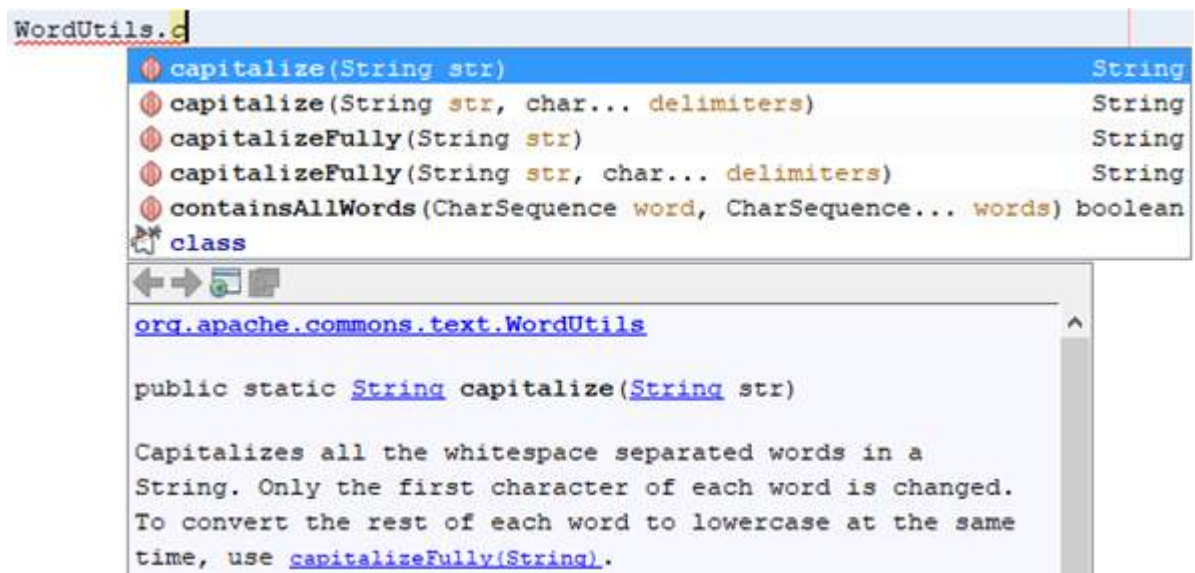


Je ziet de foutmelding Javadoc not found.

Je downloadt de documentatie die bij de library hoort:

1. Je klikt in het tabblad Projects met de rechtermuisknop op commons-text-1.2.jar
2. Je kiest Download Javadoc

Terwijl je code tikt, krijg je nu wel uitleg over de werking van classes en methods:



3.3 Version range

Je verwijst nu in `pom.xml` expliciet naar één bepaalde versie van de commons-text dependency: 1.2

Je kan ook werken met een version range (een reeks versies). Hierbij hebben vierkante haakjes de betekenis “inclusief grenswaarde” en hebben ronde haakjes de betekenis “exclusief grenswaarde”.

Voorbeelden:

- `<version>[1.0,]</version>` alle versies vanaf 1.0.
- `<version>[,2.0]</version>` alle versies tot en met 2.0.
- `<version>(1.0,]</version>` alle versies hoger dan 1.0.
- `<version>[,2.0)</version>` alle versies tot voor 2.0.
- `<version>[1.2,2.0)</version>` alle versies vanaf 1.2 tot voor 2.0.

Maven zal de versie van de dependency gebruiken die het hoogste is binnen de aangegeven range.

Bij `<version>[,2.0)</version>` zal Maven de dependency gebruiken met het hoogste volgnummer juist voor 2.0.

3.4 Scope

Je bepaalt met de scope van een dependency in welke omstandigheid de dependency ter beschikking is. Je hebt in `pom.xml` bij de dependency commons-text geen scope meegegeven. De dependency krijgt dan de standaard scope: `compile`.

De meest gebruikte scopes:

- **compile**
De dependency is beschikbaar tijdens het compileren én tijdens het uitvoeren van het project. Het JAR bestand die de dependency voorstelt kan mee verpakt worden in het eindproduct (JAR bestand, WAR bestand) van je project.
- **test**
De dependency is enkel beschikbaar tijdens het compileren en uitvoeren van geautomatiseerde testen. Het gaat om libraries die je helpen geautomatiseerde testen vlot te schrijven. Je leert hier meer over in de cursus JUnit.
- **runtime**
De dependency is niet beschikbaar tijdens het compileren, maar wel tijdens het uitvoeren en testen van het project. Het meest gebruikte voorbeeld is de dependency voor je JDBC driver. Je roept zelf de classes niet op die zich binnen in de JDBC driver library bevinden, maar de JDBC API roept wel deze classes op tijdens het uitvoeren van het project.
- **provided**
De dependency is beschikbaar tijdens het compileren.
Het JAR bestand die de dependency voorstelt wordt niet mee verpakt in het eindproduct van je project, omdat dit JAR bestand al aanwezig is in de omgeving waarin je applicatie draait. Het gaat meestal om dependencies die je oproept in een website, maar niet moeten meegeleverd worden in het WAR bestand van die website, omdat ze al aanwezig zijn in de webserver waarop de website zal draaien.

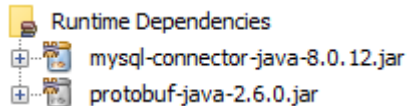
Je voegt als voorbeeld de dependency voor de MySQL JDBC driver toe aan je applicatie.

Je voegt in `pom.xml` volgende regels toe voor `</dependencies>`:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>[8,9)</version>
  <scope>runtime</scope>
</dependency>
```

Je slaat `pom.xml` en je doet een build van je project.

Je ziet in het tabblad Projects dat je project nu een onderdeel Runtime Dependencies bevat:



Je ziet dat de mysql dependency een transitive dependency heeft op de protobuf dependency.

De mysql dependency is ter beschikking gedurende de uitvoering van het project, maar niet tijdens het compileren van het project. Je ziet dit laatste op volgende manier:

1. Je klikt op het plus teken links voor mysql-connector-java-8.0.12.jar.
2. Je ziet dat de library packages bevat die beginnen met com.mysql.
3. Je tikt in de class Main het begin van een import statement:

```
package be.vdab.hallowereld;
```

```
import org.apache.commons.text.WordUtils;
import com.
public class oracle {
    public void main(String[] args) {
        System.out.println(WordUtils.capitalize("hallo"));
    }
}
```

Je ziet in het popup lijstje mysql niet vermeld, omdat deze dependency niet beschikbaar is tijdens het compileren (en dus ook het ontwikkelen) van je Java code.

4 LIFE CYCLE

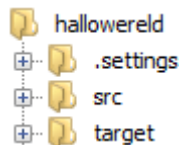
Maven heeft drie grote handelingen (life cycles) die je op je project kan uitvoeren:

- clean
- default
- site

4.1 clean

De clean life cycle verwijdt alle gecompileerde code en het eindresultaat (JAR bestand, WAR bestand) van je project.

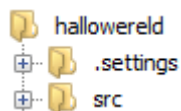
Situatie voor de clean life cycle (Files tabblad): het project bevat een map target



Je voert de clean life cycle uit:

1. Je klikt in het tabblad Projects met de rechtermuisknop op hallowereid.
2. Je kiest Clean.

Situatie na de clean life cycle (Files tabblad): de map target is verwijderd



4.2 default

De default life cycle compileert de sources, voert de geautomatiseerde testen uit en maakt een eindproduct (JAR bestand, WAR bestand) van je project.

Je voert de default life cycle uit:

1. Je klikt in het tabblad Projects met de rechtermuisknop op hallowereid.
2. Je kiest Build.

Situatie na de default life cycle (Files tabblad):

de map target is er terug en bevat onder andere hallowereid-1.0.0.jar.

Voor de volgende uitleg voer je eerst terug de clean life cycle uit.

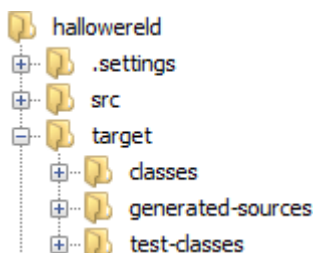
De default life cycle bestaat uit een aantal subtaken (compileren, geautomatiseerde testen uitvoeren, ...) die Maven na mekaar uitvoert.

Je kan één deeltaak afzonderlijk uitvoeren. Je voert als voorbeeld enkel de deeltaak compileren uit:

1. Je klikt in het tabblad Projects met de rechtermuisknop op hallowereid.
2. Je kiest Run Maven, Goals.
3. Je tikt compile bij Goal.
4. Je kiest OK.

Maven compileert enkel de sources en plaatst de gecompileerde versies in target/classes.

Maven maakt wel niet het eindproduct van je project.



4.3 site

De site life cycle maakt documentatie van je project.

Deze documentatie is gebaseerd op commentaar die je in de classes schrijft.

Je voegt eerst een class met zo'n commentaar toe aan je project:

```
package be.vdab.hallowereid;
/**
 * Een vierkant uit de wiskunde
 * @author Jean
 */
public class Vierkant {
    private final int zijde;
    /**
     * maakt een nieuw vierkant
     * @param zijde De zijde van het vierkant
     */
    public Vierkant(int zijde) {
        this.zijde = zijde;
    }
    /**
     * berekent de oppervlakte van het vierkant
     * @return de oppervlakte
     */
    public int oppervlakte() {
        return zijde * zijde;
    }
}
```

- (1) Je schrijft de documentatie over de class (Vierkant) als commentaar voor die class.
Je start de commentaar met `/**` en sluit hem af met `*/`. Je begint elke regel hiertussen met `*`.
- (2) Je beschrijft op de eerste regel de class.
- (3) Je tikt na `@author` de naam van de schrijver van de class.
- (4) Je beschrijft de constructor (volgend na de huidige commentaar).
- (5) Je beschrijft met `@param` de parameter van de constructor.
- (6) Je beschrijft met `@return` de returnwaarde van de method `oppervlakte`.

Je slaat de source op.

Je voert de site life cycle uit:

1. Je klikt in het tabblad Projects met de rechtermuisknop op `hallowereid`.
2. Je kiest `Generate Javadoc`.

Je ziet in het output venster dat de site life cycle HTML bestanden maakt met de documentatie van je project. Je ziet op het einde ook een hyperlink met een verwijzing naar de welkompagina van deze documentatie: [View generated javadoc at ...](#)

Als je deze hyperlink aanklikt zie je de documentatie in je browser:

All Classes
Main
Vierkant

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package be.vdab.hallowereld

Class Summary

Class	Description
Main	
Vierkant	Een vierkant uit de wiskunde

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Copyright © 2018. All rights reserved.

Als je Vierkant aanklikt, zie je de detail documentatie van deze class:

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY: NESTED | FIELD | CONSTR | METHOD](#)
[DETAIL: FIELD | CONSTR | METHOD](#)

be.vdab.hallowereld

Class Vierkant

java.lang.Object
be.vdab.hallowereld.Vierkant

public class **Vierkant**
extends Object

Een vierkant uit de wiskunde

Author:
Jean

Constructor Summary

[Constructors](#)

Constructor and Description

Vierkant (int zijde)
maakt een nieuw vierkant

Method Summary

[All Methods](#) [Instance Methods](#) [Concrete Methods](#)

Modifier and Type	Method and Description
int	oppervlakte () berekent de oppervlakte van het vierkant

5 COLOFON

Domeinexpertisemanager:	Jean Smits
Moduleverantwoordelijke:	Hans Desmet
Medewerkers:	Hans Desmet
Versie:	29/8/2018
Nummer dotatielijst:	