



Vlaamse Dienst voor


Arbeidsbemiddeling en Beroepsopleiding

Introductie tot maven

***ma*ven**

Deze cursus is eigendom van de VDAB©

Inhoudsopgave

1	MAVEN?	1
1.1	Wat is het?	1
1.2	Dependencies	1
2	JE WERKOMGEVING	2
2.1	Java, Tomcat en Eclipse	2
3	DOWNLOAD EN INSTALLATIE VAN MAVEN	3
4	CONFIGURATIE VAN DE WERKOMGEVING	4
5	MAVEN & ECLIPSE	6
5.1	Een Maven webapplicatie met Eclipse	6
5.2	De POM	8
5.2.1	Manier 1: We voegen de servlet en JSP API elk apart toe:	8
5.2.2	 Manier 2: Alles samen.	14
5.3	Welkom	14
5.4	Compilatie van het project	15
6	MEER OVER MAVEN	18
6.1	De directorystructuur	18
6.2	Plug-ins	19
6.3	Artifacts	20
6.4	Lifecycle	22
6.4.1	Clean Lifecycle	24
6.4.2	Default Lifecycle	24
6.4.3	Aanpassingen aan de default lifecycle.	26
6.4.4	Site lifecycle	28
7	SUPER POM	30
8	ADDENDUM	33
8.1	Maven gebruiken achter een proxy	33
8.2	Maven-war-plugin	35
8.3	Maven & Javadoc	35
8.4	Meer informatie	43
9	PROBLEMEN?	44
10	OEFENING	45
11	COLOFON	46

1 MAVEN?

1.1 Wat is het?

Maven is een buildtool om het dagelijkse werk van een ontwikkelaar te automatiseren. Maven is een multi-inzetbare tool voor alle aspecten van het build- en deliveryproces. Maven heeft zijn weg gevonden naar de grote Java projecten. De belangrijkste IDE's hebben ondersteuning voor Maven. Zodoende is Maven een tool waarmee, onafhankelijk van de gebruikte IDE, software gebouwd, gepackaged en gedeployed kan worden.

Ant is ook zo een tool. Ze werken beide volgens een andere filosofie. In Ant programmeer je een Ant-script waarin je stap voor stap programmeert wat er moet gebeuren, dit noem je dan "configuration over convention".

Maven werkt volgens het principe van "**convention over configuration**". Dit betekent dat heel veel zaken op voorhand zijn afgesproken en dat je dat niet telkens opnieuw tegen Maven "hoeft te vertellen". Een bijkomend voordeel is dat de basis van alle Maven-projecten op elkaar lijkt.

Dit wil onder andere zeggen dat de namen van directories in een Maven-project vooraf bepaald zijn, maar ook de volgorde van acties zoals compileren en het maken van een jar-bestand. Je kunt hier vanzelfsprekend op ingrijpen en de acties uitbreiden of wijzigen.

Het pluspunt van Maven is dat het jar-bestanden van libraries downloadt van internet samen met de afhankelijke jar-bestanden. In een configuratiebestand som je de libraries op die je wil gebruiken, Maven doet de download.

Doordat de projectstructuur bepaald is en het configuratie bestand beperkt is tot één bestand en dat bestand één naam heeft kan je een Maven project openen in elke IDE die Maven ondersteunt. Dit is belangrijk wanneer je een Maven project op een Git server deelt met collega's. Het project wordt niet vervuild met IDE afhankelijke configuratie bestanden.

1.2 Dependencies

De jar bestanden, libraries, softwarebibliotheken of hoe je ze tot nu toe noemde, noem je in een Maven project een dependency.

Een dependency kan in Maven een jar zijn die ons project nodig heeft maar ook een plug-in die de werken van Maven uitbreidt.

De ene dependency kan afhankelijk zijn van elkaar. Wanneer je een project hebt dat gebruikt maakt van Hibernate of Spring zullen deze dependencies terug afhangen van andere dependencies. Maven zal bij het downloaden van de dependency nagaan welke dependencies er nog zijn en ook deze downloaden.

Wanneer je voor de eerste keer een project compileert, kan dit even duren omdat Maven eerst alle dependencies moet downloaden van de centrale Maven repository op het internet.

Dit betekent dus dat je steeds toegang tot het internet nodig hebt wanneer je Maven projecten maakt.

2 JE WERKOMGEVING

2.1 Java, Tomcat en Eclipse

Beschik je over de laatste versie van de JDK? Bij het herwerken van deze cursus is dit **JDK-8u73**. Heb je een 64-bit besturingssysteem, dan installeer je ook de 64-bit versie van de JDK.

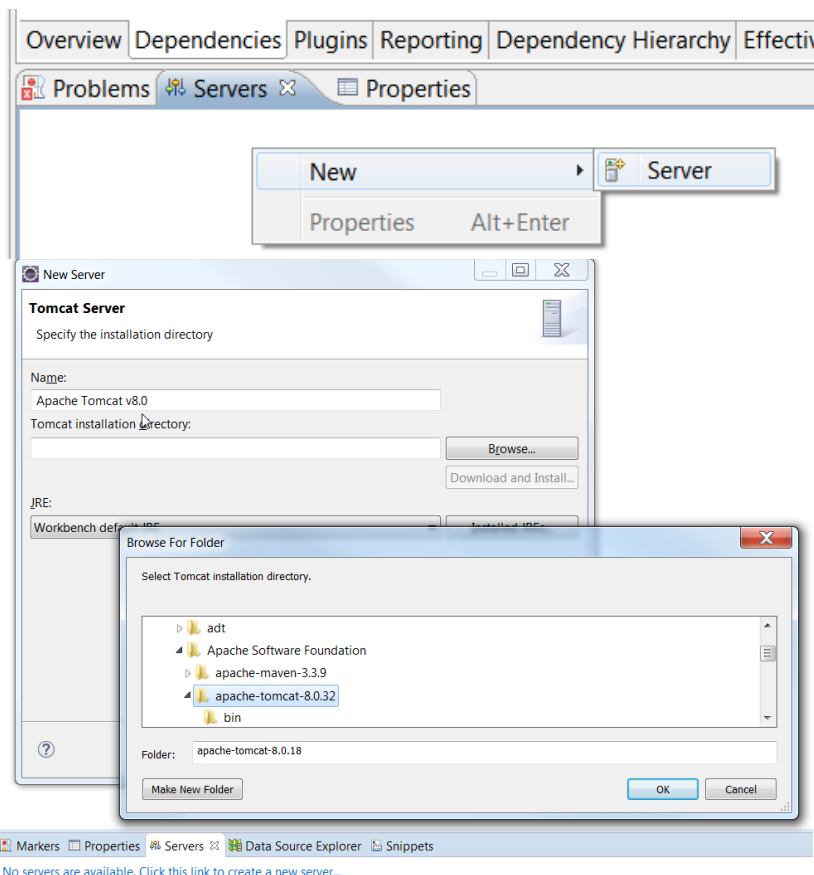
Beschik je over de laatste versie van Tomcat? Bij het schrijven van deze cursus is dit **Tomcat 8.0.32**. Ook hiervan is een 64-bit versie beschikbaar.

Installatie Tomcat: download het zip-bestand (apache-tomcat-8.0.32-windows-x64.zip) en pak het uit in bv. volgende directory C:\Program Files\Apache Software Foundation\apache-tomcat-8.0.32 .

Beschik je over de meest recente versie van Eclipse IDE voor Java EE Developers? Bij het schrijven van deze cursus is dit **Eclipse Mars**.

Installatie Eclipse: download het zip-bestand (Eclipse-jee-mars-2-win32-x86_64.zip) en pak het uit in bv. volgende directory: C:\Program Files\Eclipse.

Controleer of Eclipse verwijst naar de laatste versie van Tomcat. Wanneer Tomcat nog niet geïntegreerd is in Eclipse, klik dan rechts in het servervenster (Window-show View – Servers) en voeg een nieuwe server toe.



Kies voor Tomcat v8.0 Server en stel de locatie in van de Tomcat root-directory.

3 DOWNLOAD EN INSTALLATIE VAN MAVEN

Je downloadt Maven van <http://maven.apache.org/download.html>. Je kiest best voor de laatste stabiele versie. Voor een windows pc kies je een zip bestand en omdat je Maven niet zelf zal compileren, download je de binary versie.

The screenshot shows the Apache Maven Project website for downloading version 3.3.9. The page includes a sidebar with navigation links, a main content area with system requirements, and a table of download links.

Downloading Apache Maven 3.3.9

Apache Maven 3.3.9 is the latest release and recommended version for all users. The currently selected download mirror is <http://apache.eu.be/>. If you encounter a problem with this mirror, please select another mirror. If all mirrors are failing, there are backup mirrors (at the end of the mirrors list) that should be available. You may also consult the complete list of mirrors.

Other mirrors: <http://apache.eu.be/> [Change](#)

System Requirements

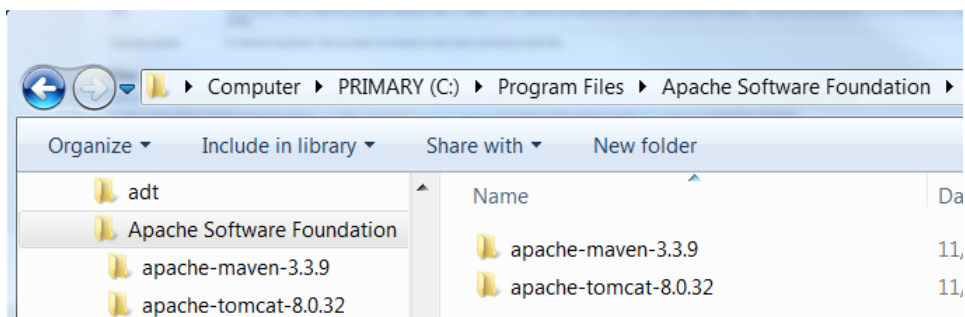
Requirement	Details
Java Development Kit (JDK)	Maven 3.3 requires JDK 1.7 or above to execute - it still allows you to build against 1.3 and other JDK versions by Using Toolchains
Memory	No minimum requirement
Disk	Approximately 10MB is required for the Maven installation itself. In addition to that, additional 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 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	Checksum	Signature
Binary tar.gz archive	apache-maven-3.3.9-bin.tar.gz	apache-maven-3.3.9-bin.tar.gz.md5	apache-maven-3.3.9-bin.tar.gz.asc
Binary zip archive	apache-maven-3.3.9-bin.zip	apache-maven-3.3.9-bin.zip.md5	apache-maven-3.3.9-bin.zip.asc
Source tar.gz archive	apache-maven-3.3.9-src.tar.gz	apache-maven-3.3.9-src.tar.gz.md5	apache-maven-3.3.9-src.tar.gz.asc
Source zip archive	apache-maven-3.3.9-src.zip	apache-maven-3.3.9-src.zip.md5	apache-maven-3.3.9-src.zip.asc


Pak het bestand uit naar een directory naar keuze. Je kan overwegen om Maven naast Tomcat te installeren in de directory "**Apache Software Foundation**".



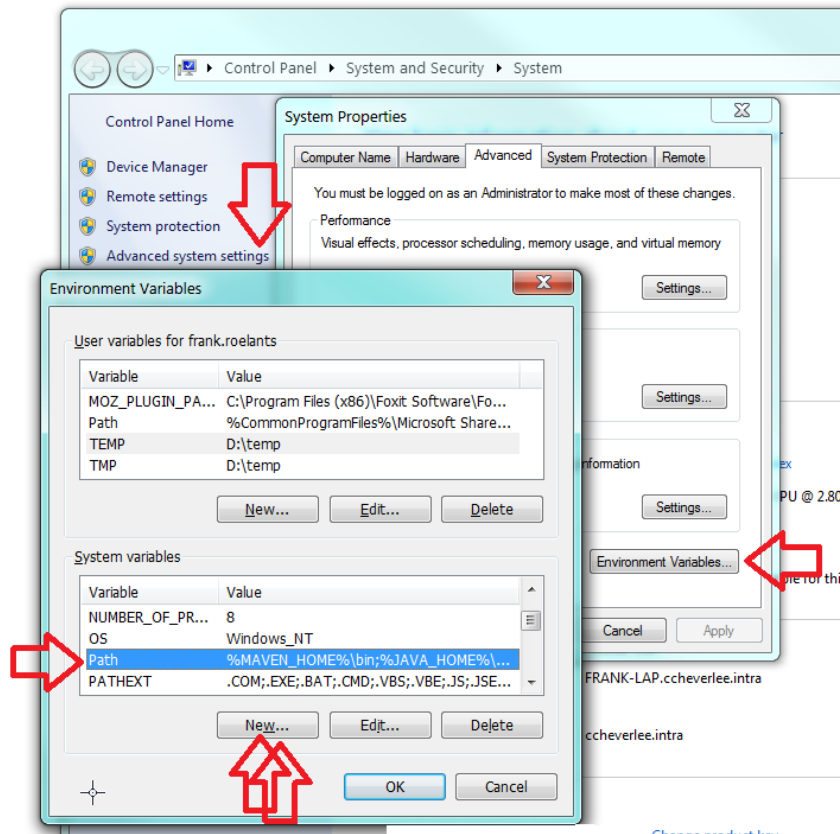
4 CONFIGURATIE VAN DE WERKOMGEVING

Neem eerst een link naar de Maven-repository <http://mvnrepository.com> op in de favorieten van je browser omdat je deze website nog frequent zult nodig hebben.

Het is een goed idee om een systeemvariabele te voorzien die de locatie van de JDK bevat. Die variabele noemt JAVA_HOME. Dit is volgens afspraak (by convention). Op een Windows 7 PC creëer je die variabele als volgt:

- Zoek in Windows Explorer +E eerst het path waar zich de JDK bevindt, klik in de adresbalk en kopieer het adres op het klembord.
Het adres is dan "**C:\Program Files\Java\jdk1.8.0_73**".
Soms zorgt een spatie in het path dat Maven niet wordt gevonden, dan kan je overschakelen op de DOS 8.3 naamgeving om de spaties te vermijden. Mocht dit bij jou het geval zijn, zoek dan de volgende woorden op je favoriete zoekmachine:
DOS + "dir/x" + windows 7
- Klik rechts op "Computer" in Windows Explorer en klik vervolgens op Properties – Advanced system settings – tabblad Advanced – knop Environment Variables... en dan bij **System variabelen** op New:
Variable name = JAVA_HOME en bij Variable path plak je het adres dat op je klembord staat. (C:\Program Files\Java\jdk1.8.0_73)
- Op dezelfde manier voeg je een systeemvariabele toe voor Maven:
Variable name = MAVEN_HOME en Variable path =
C:\Apache Software Foundation\apache-maven-3.3.9 (best te kopiëren via klembord)
- Vervolgens pas je de **systeem variabale Path** aan: Klik op de systeem variabele Path en vervolgens op de knop "Edit...".
Voeg %JAVA_HOME%\bin; **vooraan** toe aan het path. Deze verwijzing dient vooraan in het path te staan, dus zeker voor de Windows directories!

Voeg ook %MAVEN_HOME%\bin; toe.



- Sluit de vensters door op OK te klikken.

De systeem variabelen zijn aangemaakt, en Maven is toegevoegd aan het Path, dan kan je nu Maven testen.

```
C:\Windows\system32\cmd.exe

c:>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T17:41:47+01:00)
Maven home: C:\Program Files\Apache Software Foundation\apache-maven-3.3.9\bin\..
Java version: 1.8.0_73, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_73\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"

c:>_
```

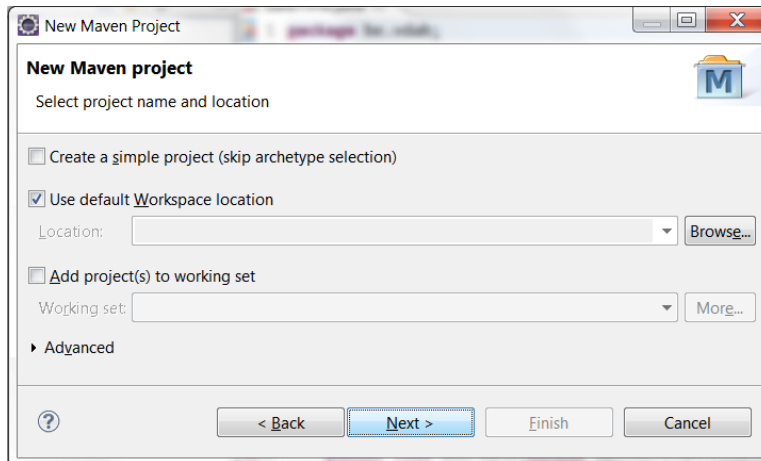
Open een command window: **Win+R** cmd en type `mvn -version`. Je ziet dan welke versie van Maven je hebt. (Wanneer je het path wijzigt moet je de command prompt herstarten.

5 MAVEN & ECLIPSE

Eclipse Mars bevat ingebouwde ondersteuning voor Maven, we hebben geen extra plugins nodig.

5.1 Een Maven webapplicatie met Eclipse

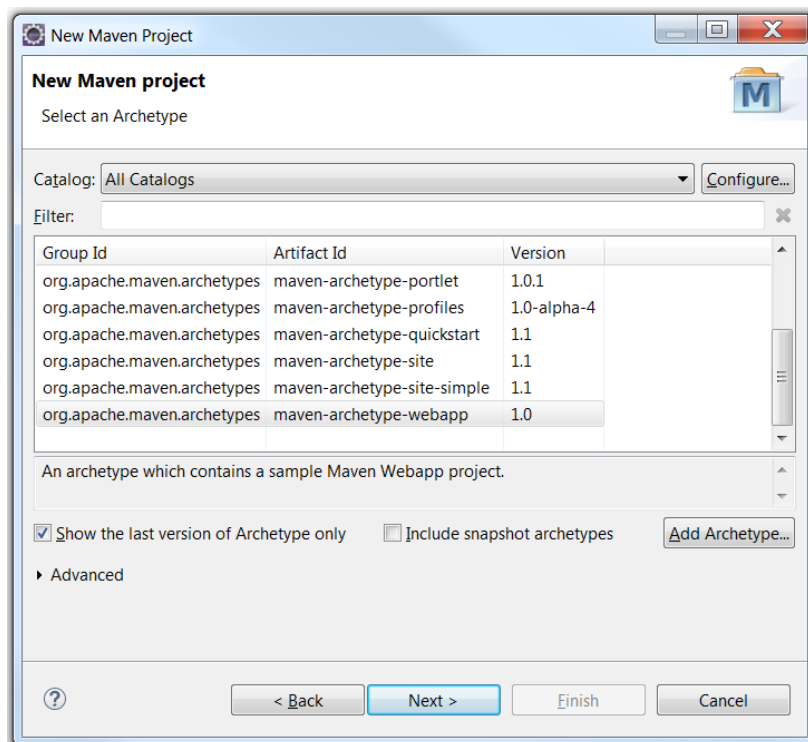
Om een nieuw project te maken, klik op File – New – Maven Project. In de volgende stap laat je alle instellingen zoals ze zijn. (tenzij je wilt afwijken van je



default workspace of je het project wil opnemen in een bestaande working set)

In de volgende stap moet je een keuze maken uit de lijst van soorten projecten, dit noemt men een archetype. De lijst is leeg maar wanneer je even wacht komt een lijst met project types te voorschijn. Rechts onderaan toont Eclipse een indicatie dat het bezig is met het opzoeken van de gegevens.

Typ *webapp* in de filter om enkel de archetypes te zien voor het bouwen van een webapplicatie. Uit de lijst selecteer je de *maven-archetype-webapp*.



Klik op Next, en vervolgens zal je het project gaan benoemen.

De naam van je project bestaat uit twee delen: het **Group Id** dat je project uniek identificeert tussen alle andere bestaande projecten. Het **Artifact Id** is de naam van het jar bestand is zonder versienummer. Meer uitleg over het artifact komt later in de cursus. Je GroupId krijgt de naam *be.vdab.centrum.voornaam.familienaam*.

Ps. vul bij centrum, voornaam en familienaam jouw opleidingscentrum, jouw voornaam en jouw familienaam in.

Opgelet: Wanneer je je naam opneemt in zowel het Group Id als het Artifact Id moet je de naam van het package aanpassen omdat de wizard het Artifact Id achter het Group Id plaatst om de package te bepalen.

The screenshot shows the 'New Maven Project' dialog box with the following fields and values:

- Group Id: be.vdab.centrum.voornaam.familienaam
- Artifact Id: cultuurhuis.voornaam.familienaam
- Version: 0.0.1-SNAPSHOT
- Package: be.vdab.centrum.voornaam.familienaam.cultuurhuis.voornaam.familienaam (highlighted with a red circle)

Below the fields is a table for 'Properties available from archetype:' with columns 'Name' and 'Value'. There are 'Add...' and 'Remove' buttons next to the table. At the bottom are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

The screenshot shows the 'New Maven Project' dialog box with the following fields and values:

- Group Id: be.vdab.centrum.voornaam.familienaam
- Artifact Id: cultuurhuis.voornaam.familienaam
- Version: 0.0.1-SNAPSHOT
- Package: be.vdab.centrum.voornaam.familienaam.cultuurhuis

Below the fields is a table for 'Properties available from archetype:' with columns 'Name' and 'Value'. There are 'Add...' and 'Remove' buttons next to the table. At the bottom are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

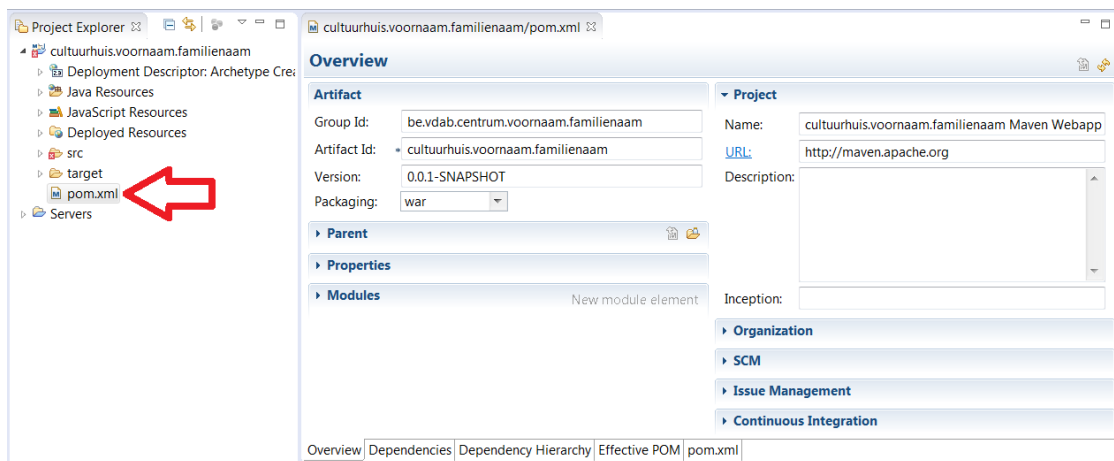
Klik op Finish en Eclipse zal het project genereren.

5.2 De POM

Het centrum van een Maven applicatie is het **pom.xml** bestand. POM is een acroniem voor Project Object Model. In de POM worden alle configuratiegegevens van het project bewaard.

Dubbelklik op *pom.xml*: de Eclipse opent een wizard, dit noemen we ook al eens de m2 wizard:

In de *Overview*-tab zie je dat het resultaat van het package proces een war bestand zal zijn.



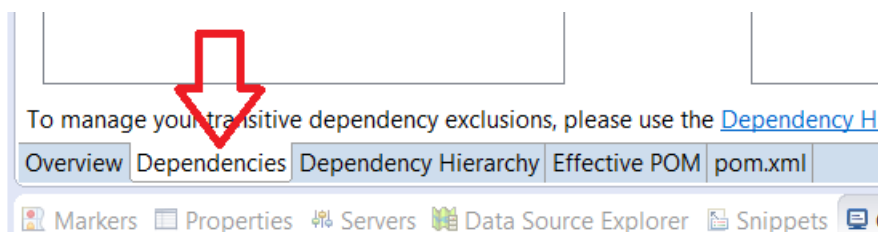
De tweede tab, onderaan het venster, *Dependencies* is een erg belangrijke tab, in volgend hoofdstuk ga je daar dieper op in. In deze tab zal je de afhankelijkheden van het project beschrijven.

Een web applicatie heeft API's nodig om servlets en JSP te kunnen gebruiken. Deze API's zal je nu aan de POM toevoegen.

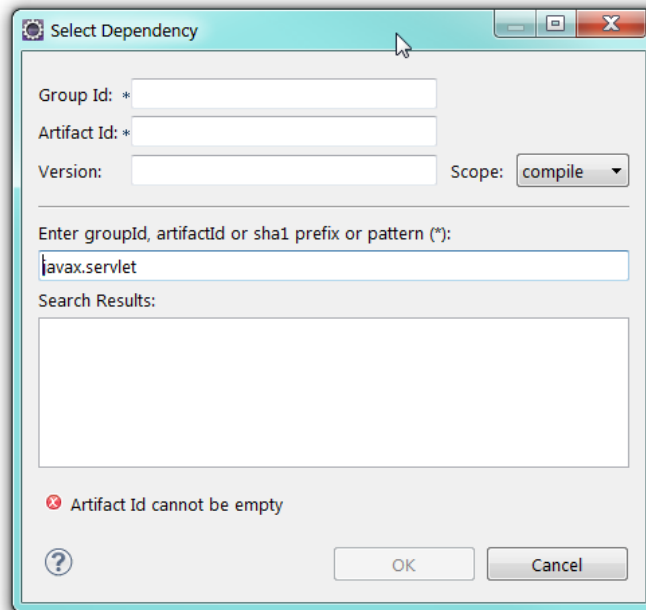
Er zijn minstens twee mogelijke manieren om de servlets en JSP API's aan je project toe voegen.

5.2.1 Manier 1: We voegen de servlet en JSP API elk apart toe:

Open de tab *Dependencies*.



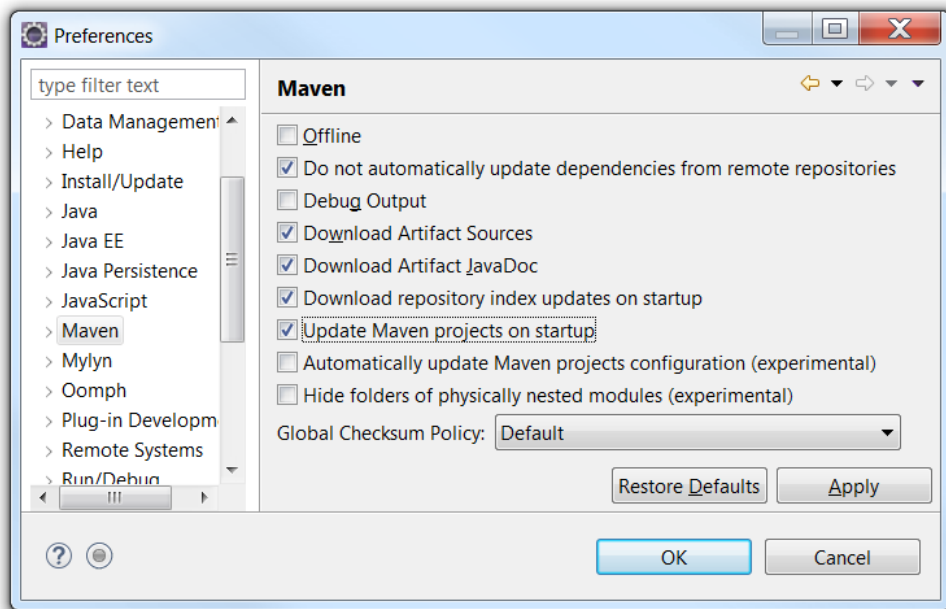
Klik daarom bij *Dependencies* op de knop *Add...* en typ in het zoekvak het volgende patroon in: **javax.servlet.***.



Als je `javax.servlet` typt zie je **dat er niets gebeurt**, dat komt doordat we nog geen overzicht hebben van de beschikbare bibliotheken. We moeten Eclipse deze eerst laten downloaden via een instelling. Klik nu op **Cancel**.

Ga naar het menu Windows – Preferences en klik in de navigatie op Maven.

Zet zeker "Download repository indexes updates en startup" aan.
Je kunt ervoor kiezen om de documentatie en de source van de API te laten

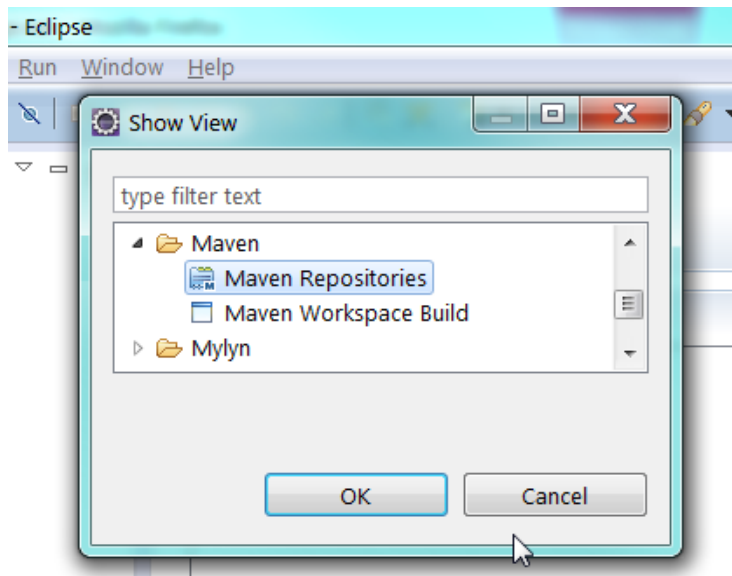


downloaden door Maven wanneer deze beschikbaar is. Wanneer je de source download kan je tijdens het debuggen daar gebruik van maken.

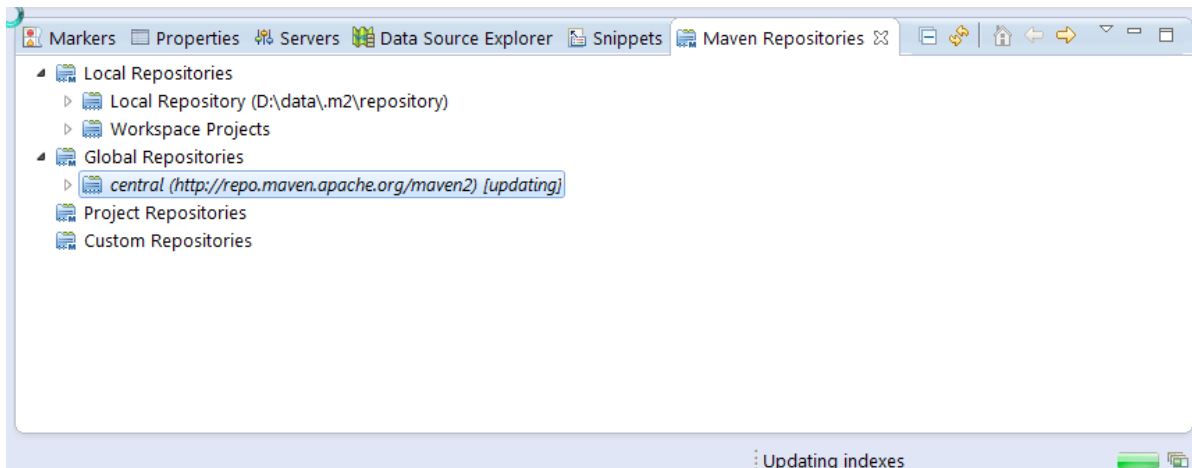
Nu moeten we Eclipse opnieuw opstarten.

Vervolgens moeten we de locale repository index opnieuw bouwen, dit kan in het venster "Maven Repositories". Standaard is dit niet zichtbaar. Je maakt het als volgt zichtbaar:

- Klik op het menu Window - Show View – Other.



- Klik Global Repositories open. Je ziet nu "central" staan. Je kan het niet open klikken omdat het nog leeg is.
- Klik rechts op central en kies uit het popupmenu "Full Index Enabled". Je ziet de muisaanwijzer wijzigen en rechts onder in de statusbar staat "updating index".



Na wat het nodige geduld (bij mij duurde het een kwartiertje) kan je de Global Repository "central" open klikken en krijg je een overzicht van de bestaande libraries.

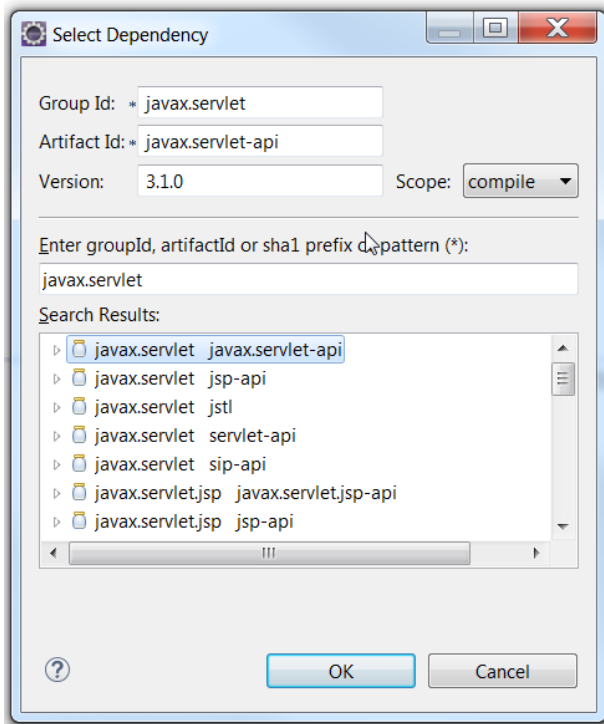


Let op, wanneer je bij de windows – preference – Maven de eerste optie "Do not automatically update dependencies from remote repositories" uitvinkt zal Eclipse bij elke opstart de dependencies bijwerken en loop je kans dat het opstarten zéér langzaam gebeurt!

Nu kunnen we terug naar het dependencies pane gaan. We klikken op add... en vullen "javax.servlet" in, in het zoekvak zien we verschillende resultaten.

Zoals je ziet zijn er meerdere artefacten waarvan het GroupId overeenkomt met javax.servlet. De vraag stelt zich dan welke moet ik kiezen? Je wilt waarschijnlijk de

laatste versie van de servlet en JSP specificatie gebruiken op Tomcat 8. Hiervoor heb je de servlet 3.1.x API's en de JSP 2.3.x API's nodig.

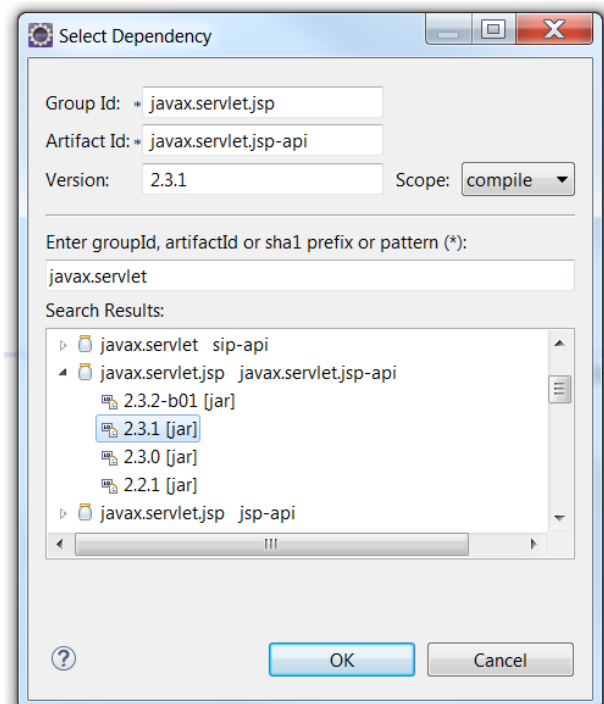


Door in de wizard het JAR bestand open te klikken kan je controleren welke versie er schuilt onder het artifactId.

Je merkt hier dat onder het artifactId **javax.servlet-api** de laatste API versie schuilt.

De JSP API's zitten onder krijgen als artifactId **javax.servlet.jsp-api**.

In de javax.servlet.jsp-api artifact zit een bèta-versie verscholen. Een beta versie herken je doordat achter het versienummer bxx staat of axx voor een alfa versie. Je vermijdt deze best in de cursus. Klik het artefact open en selecteer expliciet versie 2.3.1 maar **klik nog NIET op OK**.



Je dient de artifacts één voor één te selecteren en toe te voegen.

Verder is er nog een drop-downlist **Scope**.

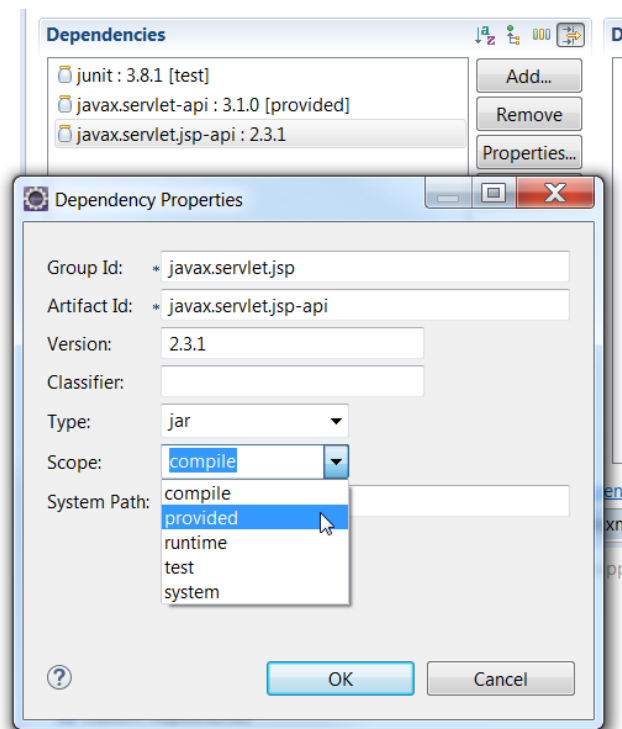
De scope plaats je op **provided**.

De “Scope” geeft aan of een JAR bestand voorzien wordt door de runtime omgeving, wat gebeurt door Tomcat in het geval van een servlet of een JSP of je het jar bestand meegeeft en dit in een test of runtime omgeving.

De mogelijke waarden voor scope zijn:

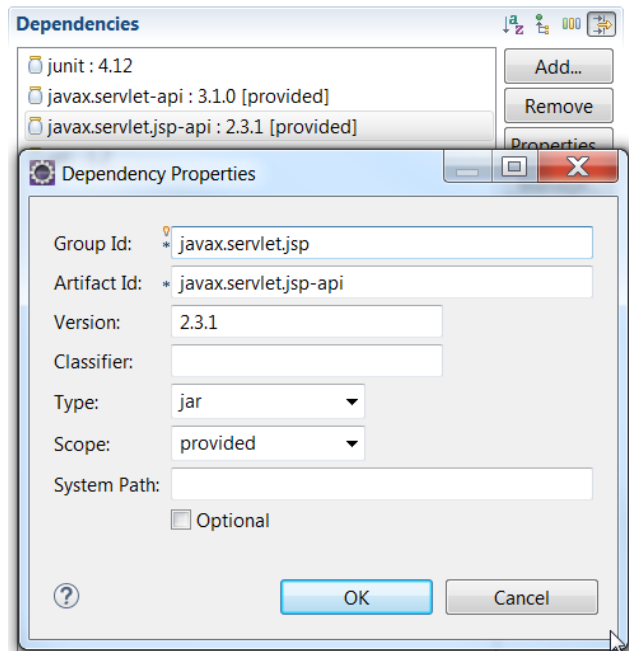
- **compile**: het jar bestand wordt opgenomen in het uiteindelijke jar/war bestand waarin de applicatie zit. Je kiest deze optie als het artefact nodig is bij compilatie maar ook tijdens de runtime en niet voorzien is door de omgeving.
- **test**: het jar bestand is alleen nodig tijdens de testfase. Het typische voorbeeld hiervan is het junit-junit artefact.
- **runtime**: het jar bestand wordt opgenomen in het uiteindelijke jar/war bestand maar is overbodig tijdens compilatie. Een typisch voorbeeld van zo’n artefact is mysql-connector-java.
- **provided**: het jar bestand wordt NIET opgenomen in het uiteindelijke jar/war bestand, want dit wordt geleverd door de runtime omgeving. De typische voorbeelden hiervan zijn de javax.servlet-api en javax.servlet.jsp-api artefacten.
- **system**: is identiek aan runtime máár het pad moet expliciet worden opgegeven.
- **import**: dit wordt gebruikt wanneer het artefact een andere POM is en dependencies moeten worden vervangen.

Je mag nu op OK klikken en Mocht je de scope niet ingesteld hebben bij het selecteren van het artifact het jar bestand toevoegen.



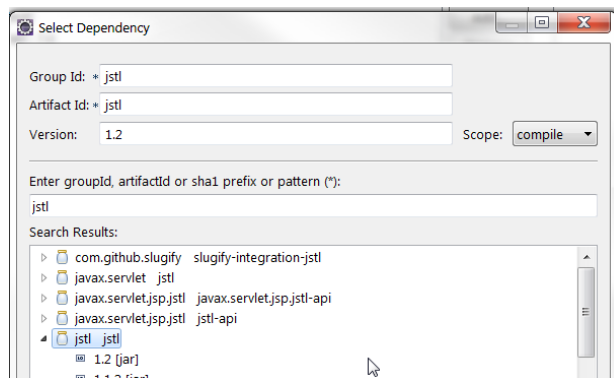
Had je toch al op OK geklikt dan kan je de scope wijzigen via de button “Properties...”.

Soms wil je niet de laatste versie gebruiken omdat je Tomcat server of een ander artifact daar niet mee kan samenwerken of omdat de laatste versie een bèta-versie is. Dan wijzig je de vereiste versie van het artefact via de “Properties...” button in het venster van de Dependency bij version. Hier is als voorbeeld de versie van de JSP API teruggebracht naar 2.3.1.



Je webapplicatie heeft nog één dependency nodig, jstl. Deze vind je terug als volgt:

- GroupId: jstl
ArtifactId: jstl
Scope: compile

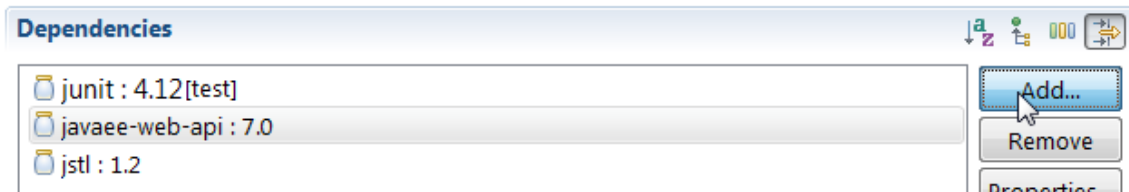


Je kan ook volgende code aan de pom.xml toevoegen:

```
<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

5.2.2 Manier 2: Alles samen.

De methode gaan we niet toepassen, je hoeft deze stap dus niet zelf te doen!



Klik daarom bij Dependencies op de knop Add... en typ in het zoekvak het volgende patroon in: javaee-web-api. In de lijst van dependencies wordt volgende dependency opgenomen.


```
<dependencies>
...
  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-web-api</artifactId>
    <version>7.0</version>
    <scope>provided</scope>
  </dependency>
...
</dependencies>
```

Je hebt nu niet alleen de Servlet 3.0 en JSP 2.2 API's opgenomen maar ook EJB Lite 3.1, JPA 2.0 EL 1.2, JSTL 1.2, JSF 2.0, JTA 1.1, JSR-45, JSR-250.

Wat moet je nu kiezen, een standaardpakket of alles individueel toevoegen. Een antwoord dat in alle situaties geldig is kan ik niet geven, het hangt al van de mate van controle die je wil over de afzonderlijke onderdelen en of je alle onderdelen gaat nodig hebben. Voor veel projecten zal het beter zijn alles apart te importeren omdat je niet altijd alle jar's nodig zal hebben.

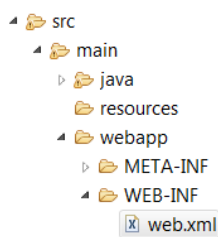
5.3 Welkom

Je moet nog aangeven welke de welcome-file zal zijn in onze applicatie.

 cultuurhuis.naam.voornaam Hiervoor open je *web.xml* en voeg je volgende code toe binnen de web-app tag:

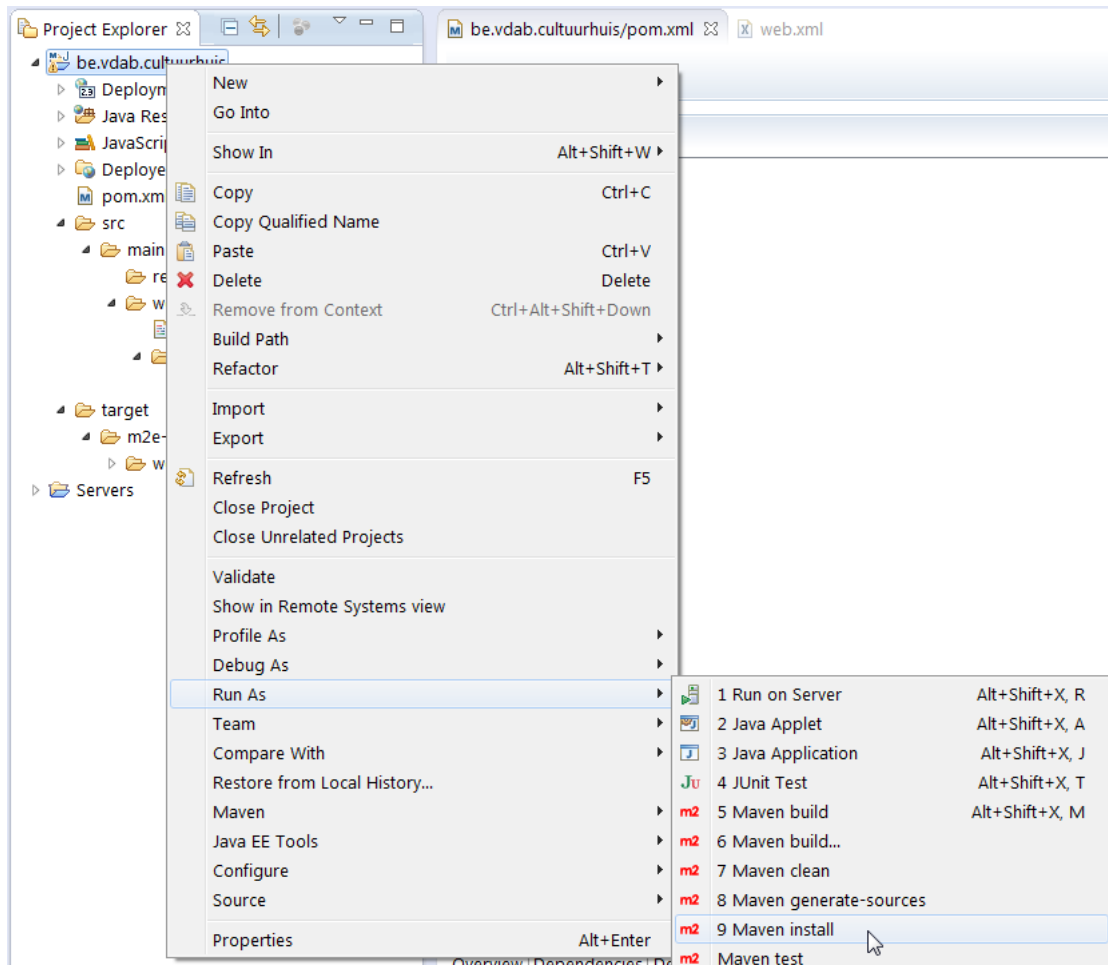
```
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

Heb je nog geen web.xml bestand, dan maak je er eentje in WEB-INF.



5.4 Compilatie van het project

Klik rechts op de naam van het project en selecteer Run As, klik vervolgens op "Maven install".



Maven schiet in actie en download artefacten, mocht je minder uitvoer hebben is dit geen probleem, onderstaande uitvoer is die van een bijna volledig project:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building be.vdab.cultuurhuis Maven Webapp 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ be.vdab.cultuurhuis ---
[INFO] execute contextualize
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.5.1:compile (default-compile) @ be.vdab.cultuurhuis ---
```



```
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-booter/2.10/surefire-booter-2.10.jar
```

```
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-api/2.10/surefire-api-2.10.jar
```



```
Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit3/2.10/surefire-junit3-2.10.jar (26 KB at 103.4 KB/sec)
```

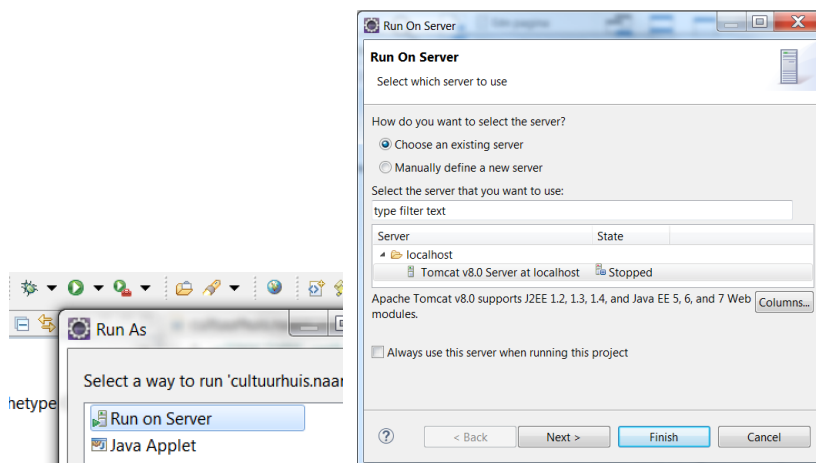
```
-----
T E S T S
-----
```

Results :

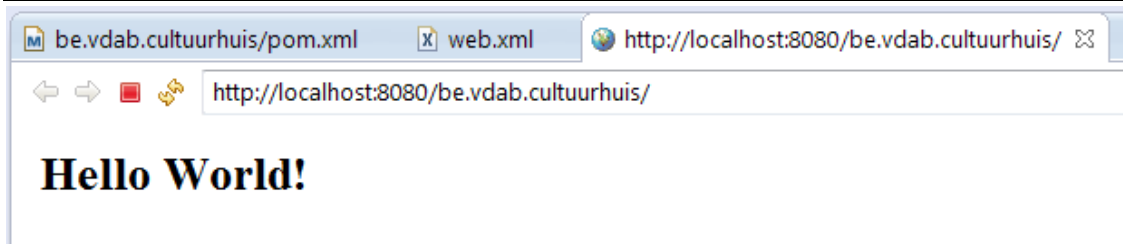
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

```
[INFO]
[INFO] --- maven-war-plugin:2.1.1:war (default-war) @ be.vdab.cultuurhuis ---
[INFO] Packaging webapp
[INFO] Assembling webapp [be.vdab.cultuurhuis] in
[D:\data\workspace_kepler\be.vdab.cultuurhuis\target\be.vdab.cultuurhuis]
[INFO] Processing war project
[INFO] Copying webapp resources [D:\data\workspace_kepler\be.vdab.cultuurhuis\src\main\webapp]
[INFO] Webapp assembled in [36 msecs]
[INFO] Building war: D:\data\workspace_kepler\be.vdab.cultuurhuis\target\be.vdab.cultuurhuis.war
[WARNING] Warning: selected war files include a WEB-INF/web.xml which will be ignored
(webxml attribute is missing from war task, or ignoreWebxml attribute is specified as 'true')
[INFO]
[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ be.vdab.cultuurhuis ---
Downloading: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.pom
Downloaded: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.pom (2 KB
at 9.5 KB/sec)
Downloading: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.7/plexus-components-
1.1.7.pom
Downloaded: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.7/plexus-components-
1.1.7.pom (5 KB at 44.2 KB/sec)
Downloading: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/1.0.8/plexus-1.0.8.pom
Downloaded: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/1.0.8/plexus-1.0.8.pom (8 KB at 46.5
KB/sec)
Downloading: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-8/plexus-
container-default-1.0-alpha-8.pom
Downloaded: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-8/plexus-
container-default-1.0-alpha-8.pom (8 KB at 36.6 KB/sec)
Downloading: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar
Downloaded: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar (12 KB
at 44.2 KB/sec)
[INFO] Installing D:\data\workspace_kepler\be.vdab.cultuurhuis\target\be.vdab.cultuurhuis.war to
D:\data\.m2\repository\be\vdab\centrum\cultuurhuis\voornaam\familienaam\be.vdab.cultuurhuis\0.0.1-
SNAPSHOT\be.vdab.cultuurhuis-0.0.1-SNAPSHOT.war
[INFO] Installing D:\data\workspace_kepler\be.vdab.cultuurhuis\pom.xml to
D:\data\.m2\repository\be\vdab\centrum\cultuurhuis\voornaam\familienaam\be.vdab.cultuurhuis\0.0.1-
SNAPSHOT\be.vdab.cultuurhuis-0.0.1-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 16.997s
[INFO] Finished at: Thu Nov 28 10:57:25 CET 2013
[INFO] Final Memory: 10M/488M
[INFO] -----
```

Vervolgens voer je het programma uit op de Tomcat server zoals je dit deed voor je Maven kende: Klik rechts op de naam van het project en selecteer Run on Server



Je web applicatie werkt.



6 MEER OVER MAVEN

6.1 De directorystructuur

Een standaard Eclipse/Netbeans/IntelliJ/... hebben elk hun eigen structuur en configuratiebestanden, hierdoor neemt de onderlinge uitwisselbaarheid af.

Maven heeft een eigen projectstructuur en die is onafhankelijk van de IDE. Het voordeel is dat je een Maven project kan openen in een IDE naar eigen keuze, voor zo verre dat er geen IDE-specifieke plugins gebruikt worden in de POM. Wanneer je in je POM de Eclipse plugin gebruikt dan kan je niet langer overschakelen naar Netbeans of IntelliJ.

src/main/java	Hierin komen je packages met je Java source bestanden.
src/main/resources	Hierin komen o.a. property bestanden, afbeeldingen, geluidsfragmenten, ... die je nodig hebt tijdens runtime. In deze directory komen de configuratiebestanden die in het classpath moeten worden opgenomen!
src/main/filters	Hier komen properties bestanden en dergelijke die tijdens compilatie aan variabelen in andere properties bestanden/POM bestanden worden toegekend.
src/main/assembly	Hier komen de bestanden die beschrijven welke bestanden op welke manier verpakt zullen worden in een archief formaat (tar, tar.gz, zip, jar, war,...) .
src/main/config	Hierin komen configuratiebestanden die NIET in het classpath moeten zitten.
src/main/scripts	Application/Library scripts voor bv bash
src/main/webapp	Web application sources, hierin komen dus de JSP pagina's, WIN-INF,... De structuur van WEB-INF blijft behouden zoals je die zag in de servlets en JSP cursus.
src/test/java	Hierin komen je packages met unit tests.
src/test/resources	Hierin komen o.a. property bestanden, afbeeldingen, geluidsfragmenten, ... die je nodig hebt tijdens het uitvoeren van de unit tests. In deze directory komen de configuratiebestanden die in het classpath moeten worden opgenomen tijdens het uitvoeren van de unit tests!
src/test/filters	Hier komen properties bestanden en dergelijke die tijdens compilatie aan variabelen in andere properties bestanden/POM bestanden worden toegekend tijdens het uitvoeren van unit tests.
src/site	In deze directory komen gegenereerde HTML pagina's met test rapporten, documentatie ,...

Het is belangrijk dat je de structuur van Maven respecteert. Wanneer je bestanden niet de plaats zet waar Maven ze verwacht, zal je project niet functioneren.

6.2 Plug-ins

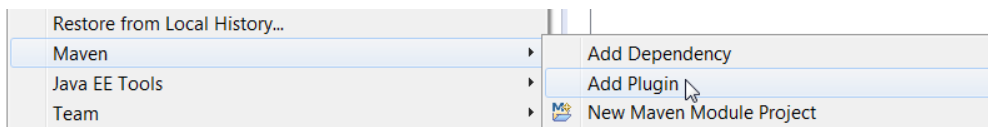
Je kan de werking van Maven uitbreiden door plug-ins toe te voegen. Hiermee kan je allerlei tools opstarten vanuit Maven. De plug-ins zijn specifiek voor dit project! Wil je dezelfde plug-ins gebruiken in een ander project kan je de code kopiëren, er bestaan complexere manieren om delen van een pom te hergebruiken, maar deze vallen buiten de doelstelling van deze cursus. Een overzicht van de plugin's vind je op mavencentral <http://mvnrepository.com/plugins.html>.

Een belangrijke plug-in is de Maven-compiler-plugin.

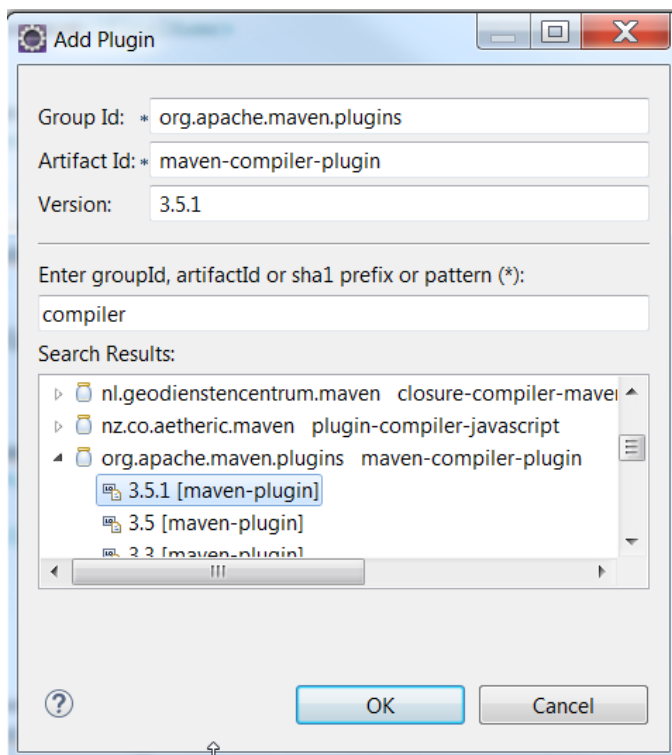
Maven compileert Java code volgens de instelling voor Java 1.5!

Wanneer je gebruik maakt van de mogelijkheden van Java 6 en hoger in de source code is er een plug-in nodig.

Je voegt deze plug-in als volgt toe: klik rechts op je project naam, uit het contextmenu kies je nu Maven – Add Plugin



Vervolgens zoek je de plugin, wij hebben de maven-compiler-plugin nodig. Je kan de naam volledig of een deel ervan intypen. Wanneer je een deel van de naam intypt wordt de lijst met zoekresultaten groter. Hoe weet je welke je nodig hebt? Dikwijls zijn er specifieke varianten voor bv de Rhino compiler (Javascript compiler die op de JVM draait), wij hebben die van apache nodig, wanneer we die open



klikken zien we alle versies, we kiezen met nieuwste stabiele versie (geen beta versie of een release candidate) om al teveel problemen te vermijden.

Je configureert de plug-in vanuit de xml weergave. Open deze view door te klikken op `pom.xml`.

Voeg in de body van de *plugin* tag de configuratie toe waarmee je de source en target versie van de Java source bepaalt:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.5.1</version>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
  </configuration>
</plugin>
```

Wanneer je in de Java source gebruik maakt van typische Java 6 of Java 7 code zal deze code nu probleemloos compileren.

De lijst van geïnstalleerde plugins is alleen te overzien in de xml code en is momenteel geen grafisch overzicht van de geïnstalleerde plugins.

6.3 Artifacts

Allereerst even een verduidelijking:


Artifact is een Engels woord, in het Nederlands wordt dat artefact. Om vergissingen te vermijden wordt in de cursus het Engels woord artifact gebruikt.

Volgens Vandale heeft artefact twee betekenissen:

- Een opzettelijk vervaardigd voorwerp
- Een door ongeldige redeneren verkregen onderzoeksresultaat.

Het artifact in de cursus is dus een opzettelijk vervaardigd “voorwerp” (software) dat je nodig hebt om je eigenlijke “voorwerp” (software) te kunnen maken.

Een artifact heeft naast een groupId en artifactId een versienummer. Je kunt in de *pom.xml* het versienummer opgeven.

Voor de dependencies die je al toegevoegd hebt is dat reeds gebeurd. Wanneer je de POM opent en klikt op  dan krijg je de xml weergave te zien. De dependency voor javax.servlet.jsp-api ziet er als volgt uit:

```
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>javax.servlet.jsp-api</artifactId>
  <version>2.3.1</version>
  <scope>provided</scope>
</dependency>
```

Wanneer het belangrijk is dat een bepaalde versie van de dependency gebruikt wordt noteer je die als volgt: `<version>2.3.1</version>`.

Maar je hebt ook de mogelijkheid om een minimale en/of maximale waarde op te geven. Enkele voorbeelden:

- | | |
|---|--|
| • <code><version>[2.0,]</version></code> | alle versies vanaf 2.0 |
| • <code><version>[,2.0]</version></code> | alle versies tot en met 2.0 |
| • <code><version>(2.0,]</version></code> | alle versies hoger dan 2.0 |
| • <code><version>[,2.0)</version></code> | alle versies tot vóór 2.0 |
| • <code><version>[2.0,2.2.1]</version></code> | vanaf 2.0 tot en met 2.2.1 |

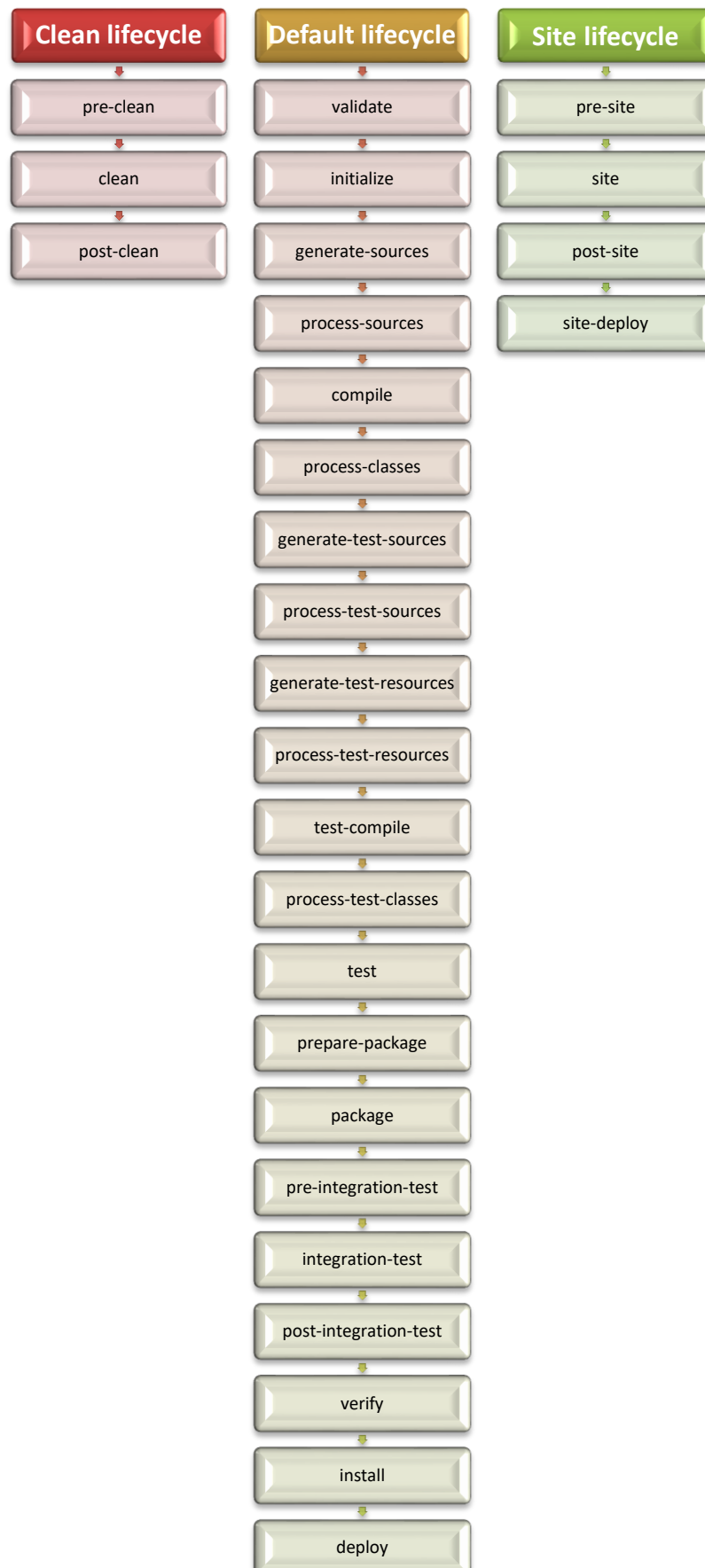
Vierkante haakjes [] betekenen **inclusief de grenswaarde**.

Ronde haakjes () betekenen **exclusief de grenswaarde**.

De haakjes zijn combineerbaar, tussen de haakjes bevindt zich minstens één versienummer en een komma.

6.4 Lifecycle

Maven kent drie lifecycles waarbij elke lifecycle uit meerdere fases bestaat.

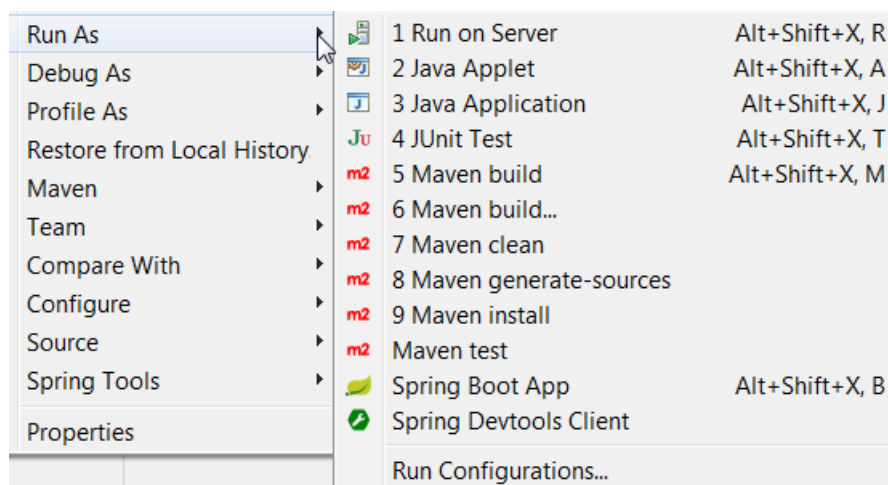


Elke lifecycle start steeds bij de eerste fase en kan worden uitgevoerd tot een bepaalde fase.

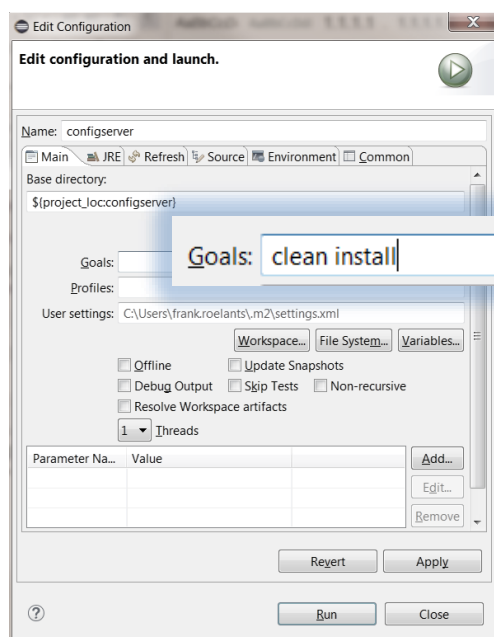
Dus wanneer je in het menu kiest voor “Run As 9 Maven install” dan worden alle fases uit de default cyclus uitgevoerd tot en met de install fase. De fase tot waar je de lifecycle uitvoert noem je de goal. Je kunt ingrijpen in elke fase. Maven kent naast dependencies ook plug-ins die je kunt laten uitvoeren in een bepaalde fase. Maven zal een aantal plug-ins uitvoeren in een standaard lifecycle. Je kunt dit bekijken wanneer je de effective pom opvraagt. Klik hiervoor op de tab “Effective POM”. Effective POM is een read-only voorstelling van de pom met daarin alle gegenereerde afhankelijkheden.

Elk van deze plug-ins kan je ook zelf aan je eigen POM toevoegen en daardoor kan je ingrijpen op hun werking door de configuratie aan te passen.

In het Run As menu vinden we een aantal Maven opties terug, de meeste zijn shortcuts naar een bepaalde goal.



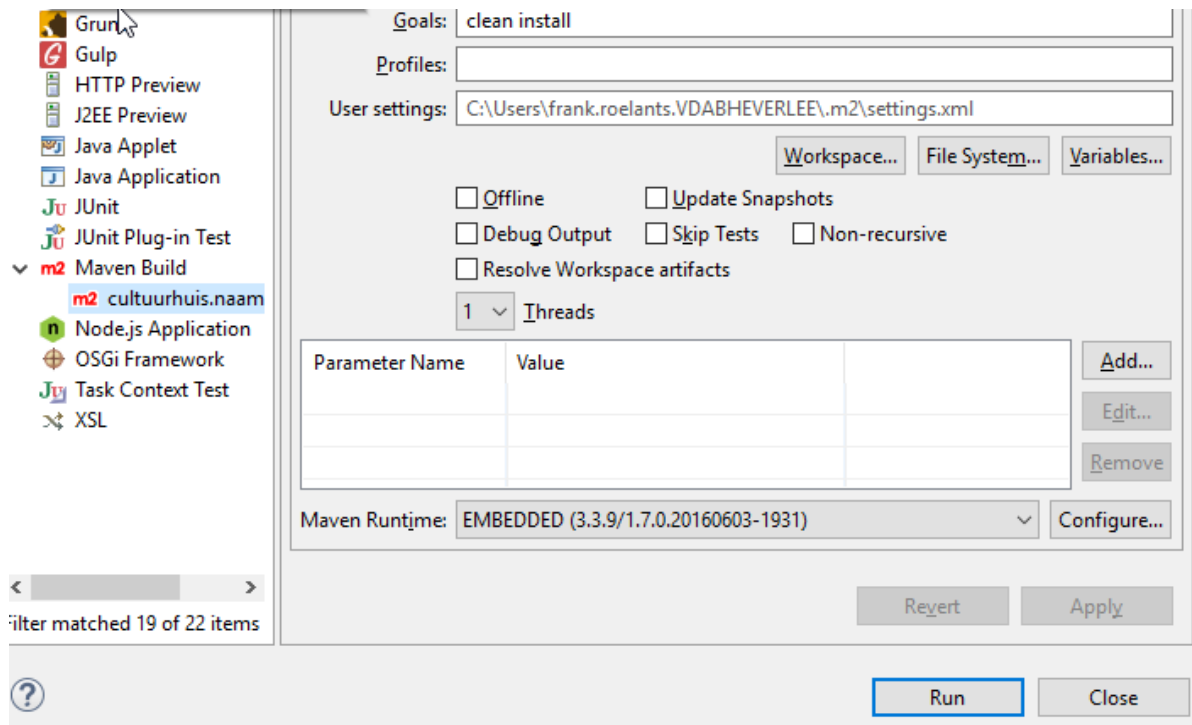
De optie “5 Maven build...” biedt de mogelijkheid om zelf één of meerdere goals door spaties van elkaar te scheiden. De goals sommen we op in het veld Goals.



Denk er aan dat als je een bepaalde goal wil uitvoeren steeds alle voorgaande coals uit dezelfde lifecycle worden uitgevoerd!

Wanneer je de “clean install” goals opnieuw wil laten uitvoeren kies die voor Run As

– Run Configurations... en dan kan je terug kiezen voor de bestaande run configuratie “clean install”



6.4.1 Clean Lifecycle

In deze lifecycle worden alle bestanden, die gemaakt werden tijdens de vorige build, verwijderd. Hierdoor worden de gecompileerde classes, de gegenereerde documentatie, de jar bestanden allemaal verwijderd.

Je voert de lifecycle uit door rechts te klikken op het project en te kiezen voor *Run As* vervolgens *Maven clean*.

pre-clean	<i>Voert processen uit die moeten plaatsvinden voor de eigenlijke opkuis.</i>
------------------	---

clean	Alle bestanden die gemaakt werden tijdens de vorige build worden verwijderd.
--------------	--

post-clean	<i>Voert processen uit die moeten plaatsvinden na de eigenlijke opkuis.</i>
-------------------	---

In een standaard POM is alleen de clean fase(goal) van de clean lifecycle gekoppeld aan de maven-clean-plugin. Meer uitleg over allerlei configuratiemogelijkheden van deze plug-in vind je hier:

<http://maven.apache.org/plugins/maven-clean-plugin/>

6.4.2 Default Lifecycle

In de default lifecycle gebruik je onder meer het project compileren, testen, code opladen naar een versiebeheersysteem, het project omzetten tot een jar/war en deze eventueel deployen op een server.

Je voert een goal uit deze lifecycle uit, door rechts te klikken op het project en te kiezen voor *Run As*, *Maven xxx*, waarbij xxx één van onderstaande goals is. Merk op dat niet elke goal tussen de snelkeuze lijst staat.

<code>validate</code>	Valideer het project en controleer of alle informatie aanwezig is.
<code>initialize</code>	Initialiseer de build-fase door properties klaar te zetten en directories te creëren.
<code>generate-sources</code>	Genereer broncode die nodig is voor compilatie.
<code>process-sources</code>	Verwerk de broncode, hier kunnen filters toegepast worden.
<code>generate-resources</code>	Genereer resources.
<code>process-resources</code>	Kopieer de resources naar de target directory, en bereid ze eventueel voor om in te pakken.
<code>compile</code>	Compileer de broncode van het project.
<code>process-classes</code>	Voer een eventuele nabewerking uit op de gecompileerde class bestanden.
<code>generate-test-sources</code>	Genereer test classes die nodig zijn bij het testen.
<code>process-test-sources</code>	Verwerk de broncode van de test classes, hier kunnen filters toegepast worden.
<code>generate-test-resources</code>	Genereer resources die nodig zijn bij het testen van de classes.
<code>process-test-resources</code>	Kopieer de resources naar de test directories.
<code>test-compile</code>	Compileer de test classes.
<code>process-test-classes</code>	Voer een eventuele nabewerking uit op de gecompileerde test class bestanden. Deze fase is er pas vanaf Maven 2.0.5.
<code>test</code>	Voer de testen uit, maar deze testen mogen niet vereisen dat de classes verpakt zijn in een jar/war/ear/... bestand noch dat ze gedeployd zijn.
<code>prepare-package</code>	Voor stappen uit die het verpakken van de classes tot een jar/war/ear/... voorbereiden. Deze fase is er pas vanaf Maven 2.1.
<code>package</code>	Verpak de te distribueren bestanden tot bijv. een jar of war bestand.
<code>pre-integration-test</code>	Voer de stappen uit om de integratie testen voor te bereiden.
<code>integration-test</code>	Verwerk en deploy het jar/war bestand naar een omgeving waarin de integratie testen plaatsvinden.
<code>post-integration-test</code>	Kuis de integratie test omgeving op na het uitvoeren van de integratie testen.
<code>verify</code>	Controleer de geldigheid van het package en ga de kwaliteitseisen na.
<code>install</code>	Installeer het package (jar/war/ear) in de lokale repository, zodat het beschikbaar is als dependency voor andere projecten.
<code>deploy</code>	Deze goal vindt plaats bij de uiteindelijke oplevering van het project, het project wordt overgedragen naar de productieomgeving.

De meest gebruikte goal uit de default lifecycle is package. Hierdoor wordt al jouw werk gecompileerd en omgezet in een jar/war/ear bestand. De volgende stappen zijn vooral bedoeld om in team aan een project te werken.

6.4.3 Aanpassingen aan de default lifecycle.

6.4.3.1 Nieuw gedrag toevoegen.

Het gebeurt regelmatig dat de nood ontstaat om project directories op te kuisen voor een nieuwe compilatie. Dit kan je automatiseren door in te grijpen in bovenstaande lifecycle. Je kan er voor zorgen dat de “clean goal” van de clean lifecycle wordt uitgevoerd gedurende de initialize fase van de default lifecycle. Hiervoor open je de POM file in xml weergave.

Je voegt onderstaande code toe aan de body van de build tag. Je zegt in je POM dat je de maven-clean-plugin wil gebruiken en je geeft op welke versie je wil gebruiken. Hier zou je ook kunnen gebruik maken van de notatie met de haakjes en [2.4.1,] schrijven. In de executions tag plaats je één execution. Je wilt dat de plugin zijn goal uitvoert tijdens de initialize fase. Het id van de execution mag je vrij kiezen.

```
<plugins>
.
.
<plugin>
  <artifactId>maven-clean-plugin</artifactId>
  <version>2.4.1</version>
  <executions>
    <execution>
      <id>auto-clean</id>
      <phase>initialize</phase>
      <goals>
        <goal>clean</goal>
      </goals>
    </execution>
  </executions>
</plugin>
.
.
</plugins>
```

Wanneer je nu rechts klikt op het project en vervolgens klikt op Run As Maven install zie je dat clean werd uitgevoerd.

Je vindt meer informatie over deze plug-in op <https://maven.apache.org/plugins/maven-clean-plugin/>.



Merk op dat Eclipse een fout aangeeft wanneer je bovenstaande de clean plugin wil laten uitvoeren vanuit de initialize fase.

Maven Problems (1 item)
Plugin execution not covered by lifecycle configuration: org.apache.maven.plugins:maven-clean-plugin:3.0.0:clean (execution: auto-clean, phase: initialize)

Maar het voorbeeld is afkomstig van de documentatie van de maven-clean-plugin:

<https://maven.apache.org/plugins/maven-clean-plugin/usage.html> en het werkt. Wanneer je hetzelfde project met Netbeans opent dan krijg je geen fout op de execution fase van de maven-clean-plugin.



Je verwijdt nu volledig alle code van de clean plug-in uit pom.xml.

Zoals we op het einde van hoofdstuk 6.4 Lifecycle zagen kunnen we twee of meer goals bv "clean install" na elkaar laten uitvoeren. Dit is zuiverder en dan zal niet steeds de Maven? clean goal worden uitgevoerd als alleen een goal uit de default lifecycle moet worden uitgevoerd.

6.4.3.2 Gedrag wijzigen

Wanneer je werkt volgens Test Driven Development, dan heb je eerst testen geschreven en heb je nog geen code of onvolledige code. Een aantal testen zullen falen en Maven stopt de huidige test cyclus. Dat is ongewenst gedrag, we willen alle testen laten doorlopen ook al falen er enkele.

De plug-in die verantwoordelijk is voor het uitvoeren van de **JUnit testen** is de **maven-surefire-plugin**.

Je kan deze terugvinden wanneer je in de m2 wizard (zie 5.2 De POM, p8) van Eclipse onderaan op de tab "effective POM" klikt.

```
<plugins>
.
.
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
</plugin>
.
.
</plugins>
```

In de effective POM had de plug-in noch *configuration* tag noch een version tag. Door de plug-in op te nemen in onze eigen POM kunnen we het ongewenste gedrag bijsturen en weten we met zekerheid welke versie we gebruiken door de tag te voorzien van volgende configuratie:

```
<plugins>
.
.
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.19.1</version>
  <configuration>
    <testFailureIgnore>true</testFailureIgnore>
  </configuration>
</plugin>
.
```

</plugins>

Je vindt meer informatie over deze plug-in op:

<https://maven.apache.org/plugins/maven-surefire-plugin/>

6.4.4 Site lifecycle

In deze lifecycle wordt een site gegenereerd met de projectdocumentatie.

Dit kan de API-documentatie van je project zijn op basis van de javadoc.

In deze fase kunnen ook de JUnit test rapporten verzamelen en omzetten in HTML.

Je kan aan de site documentatie toevoegen die toont hoeveel van de code getest is tijdens de JUnit testen. Dit noem je de code/test-coverage. De plug-in die het coverage rapport opstelt voorziet je code van een achtergrondkleur: groen betekent dat de code gebruikt werd in een test, rood betekent ongebruikt in een test.

Let op! je krijgt slechts een beperkt beeld krijgen van de kwaliteit van je testen.

Wanneer je een test schrijft voor een constructor die op zijn beurt een setter oproept dan wordt de setter doorlopen maar deze is daarmee nog niet apart getest!

Wanneer je niet alleen werkt zijn afspraken over codering een absolute noodzaak. Om de afspraken te controleren, bestaan er ook plug-ins en hun rapporten kan je ook in deze fase opnemen in de project documentatie.

In het addendum over de maven-site-plugin wordt iets dieper op deze mogelijkheden in gegaan.

De site lifecycle bestaat uit volgende fases:

pre-site	Voert processen uit nodig om de generatie van de projectdocumentatie mogelijk te maken.
site	Genereert de projectdocumentatie.
post-site	Voert processen uit die de projectdocumentatie finaliseren en de deploy voorbereiden.
site-deploy	Deployt de gegenereerde documentatie naar de website.

Er zijn twee werkwijzen waarmee we een site kunnen bouwen.

- Een eerste manier is via de maven-site-plugin.
We kunnen er voor zorgen dat de maven-site-plugin automatisch wordt uitgevoerd wanneer de code is gecompileerd en de testen hebben plaats gevonden, hiervoor configureren we de plugin als volgt:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-site-plugin</artifactId>
  <version>3.5</version>
</plugin>
```

Het aanmaken van de site kunnen we via een executions tag laten uitvoeren vanuit de default life cycle. Maar meestal is het onnodig om bij elke compilatie alle mogelijke rapporten te laten genereren. Deze manier om een site te bouwen zal waarschijnlijk weinig gebruikt worden.

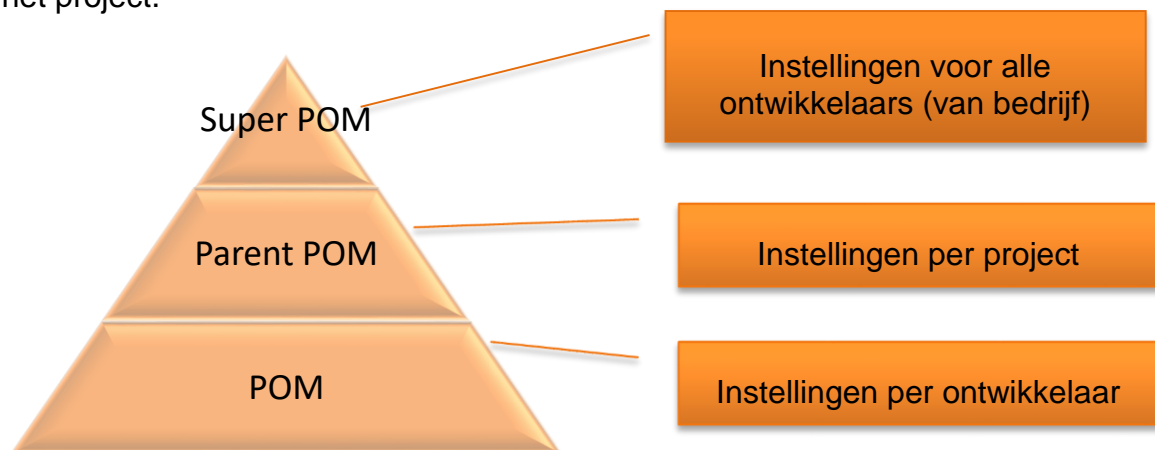
- De tweede manier bestaat er uit de plugins die de site opbouwen op te nemen in de reporting tag die komt na de </build> tag.vb:
We voeren de site cyclus nu uit zoals we op het einde van hoofdstuk 6.4 Lifecycle zagen maar als goal schrijven we nu “site”.

```
</build>
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>2.9</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>2.10.4</version>
        <configuration>
          <outputDirectory>
            ${project.build.directory}/apidocs
          </outputDirectory>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>cobertura-maven-plugin</artifactId>
        <version>2.6</version>
        <configuration>
          <formats>
            <format>html</format>
            <format>xml</format>
          </formats>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
```

7 SUPER POM

Dit is geen verhaal over de zoveelste superheld maar een verhaal over hoe je Maven gebruikt in een groter project.

Een POM kan instellingen, dependencies, plug-ins overerven van een andere POM. Hierdoor kan een POM gecreëerd worden op het niveau van het bedrijf, waar dan de versies van gekochte artifacts worden in vastgelegd. Versies van artifacts worden ook gelimiteerd omdat de gebruikte servers geen hogere versies kunnen ondersteunen. Er wordt vrijwel altijd nog een POM gemaakt op het niveau van het project waarin de afspraken en instellingen voor het project worden bewaard. En vervolgens heb je dan je eigen POM met de instellingen voor je eigen taak binnen het project.



Wanneer de versienummers worden opgegeven in een POM waar je eigen POM van erft, moet je natuurlijk zelf geen versienummers opgeven.

In de super of parent POM kan naast de versies van de dependencies ook de instellingen van plug-ins worden opgenomen. Zo kan de versie van Java worden bepaald met de maven-compiler-plugin. Zo kan er per project bepaald worden welke repository zal gebruikt worden voor het versiebeheer.

Je laat je POM bestand als volgt erven van een parent POM:

```
<parent>
  <groupId>be.vdab.parentalpom</groupId>
  <artifactId>SpringHibernatePom</artifactId>
  <version>1.2.0</version>
</parent>
```

De <parent> tag neem je in je pom file op net onder de tag <project>.

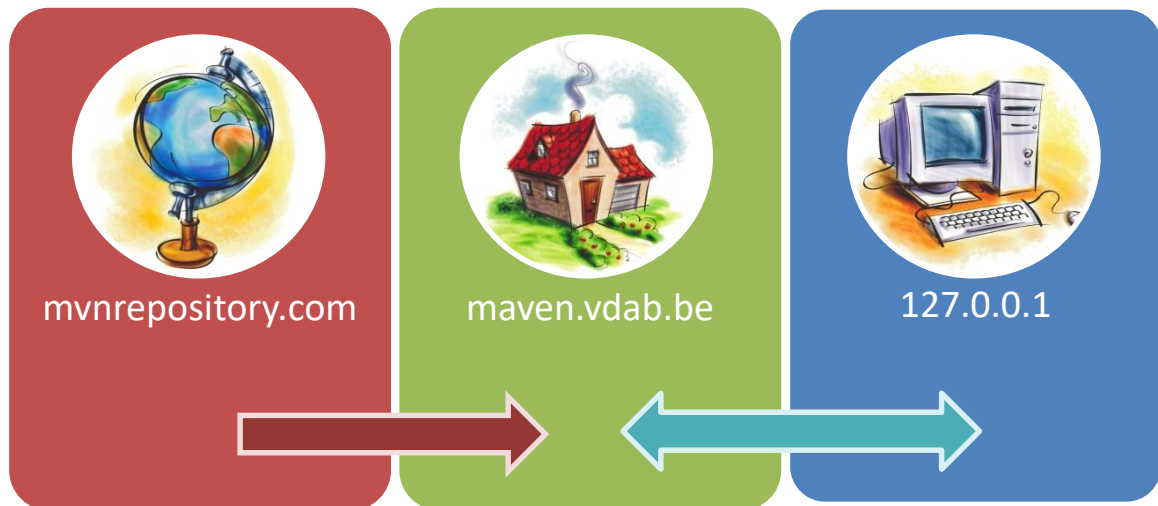
Het opstellen van een super/parent POM valt buiten het bestek van deze cursus. Meer informatie over dit onderwerp vind je op deze sites:

- <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>
- <http://books.sonatype.com/mvnref-book/reference/>

Je weet nu hoe je een POM kan afleiden van een andere POM, maar wat doe je met de dependencies/plug-ins zelf? Stel dat je het als ontwikkelaar wel handig zou vinden om een bepaalde library te gebruiken maar ze is niet voorzien in de parent-/super-POM dan kan je die toevoegen aan je eigen POM en zal Maven de files van internet downloaden. De gedownloade dependencies komen terecht in de local repository. Deze bevindt zich op je eigen pc.

Wanneer je met Eclipse werkt en de m2 plug-in is geïnstalleerd, dan staat de locatie van je local repository in het de installatiedirectory van Maven, bij mij is dit in : *apache-maven-3.3.9\conf/settings.xml*.

Java ontwikkeling gebeurt meestal in teamverband en dan is het een slecht idee om zonder overleg een dependency toe te voegen aan je project. Om dit te



voorkomen kan de toegang tot de centrale Maven repository worden verboden in de proxyserver en zet men in het bedrijf een eigen repository server op.

Dit heeft een dubbel voordeel. De bedrijfsrepository bevat enkel de dependencies die mogen gebruikt worden en het is mogelijk om de beschikbare versies te beperken. Het neveneffect hiervan is dat het internetverkeer hierdoor wordt beperkt doordat niet elke ontwikkelaar apart zijn dependencies van het internet haalt. Gekochte libraries kunnen langs deze weg worden verspreid binnen het bedrijf. Je eigen werk dat klaar is om in productie te gaan kan via de Maven deploy opdracht op de bedrijfs repository worden geplaatst.

Wanneer het bedrijf of het team een repository heeft moet je dit opgeven in je POM bestand, meestal zal dit gebeuren in de parent- of super-POM omdat dit voor alle ontwikkelaars identiek is. Hoe je in een POM naar een bedrijfs-repository verwijst valt buiten het bestek van deze cursus.

Wanneer je werkt met een super-pom dan plaats je in die super-pom al je plug-ins die je ook in de onderliggende projecten wil hebben in een tag `pluginManagement`, de onderliggende projecten erven dan deze plug-ins

```
<pluginManagement>
  <plugins>
    ...
  </plugins>
</pluginManagement>
```

Wanneer je bij je plug-ins die je vermeldt in je `pluginManagement` een versienummer opgeeft dan zal Maven ervoor zorgen dat je dependencies en de dependencies daarvan een dezelfde versie krijgen zodat geen tegenstrijdigheden ontstaan.

Voor grote projecten zoals spring bestaat daar een aparte dependency voor, dit noemt men een Bill Of Materials of een BOM.

bv:

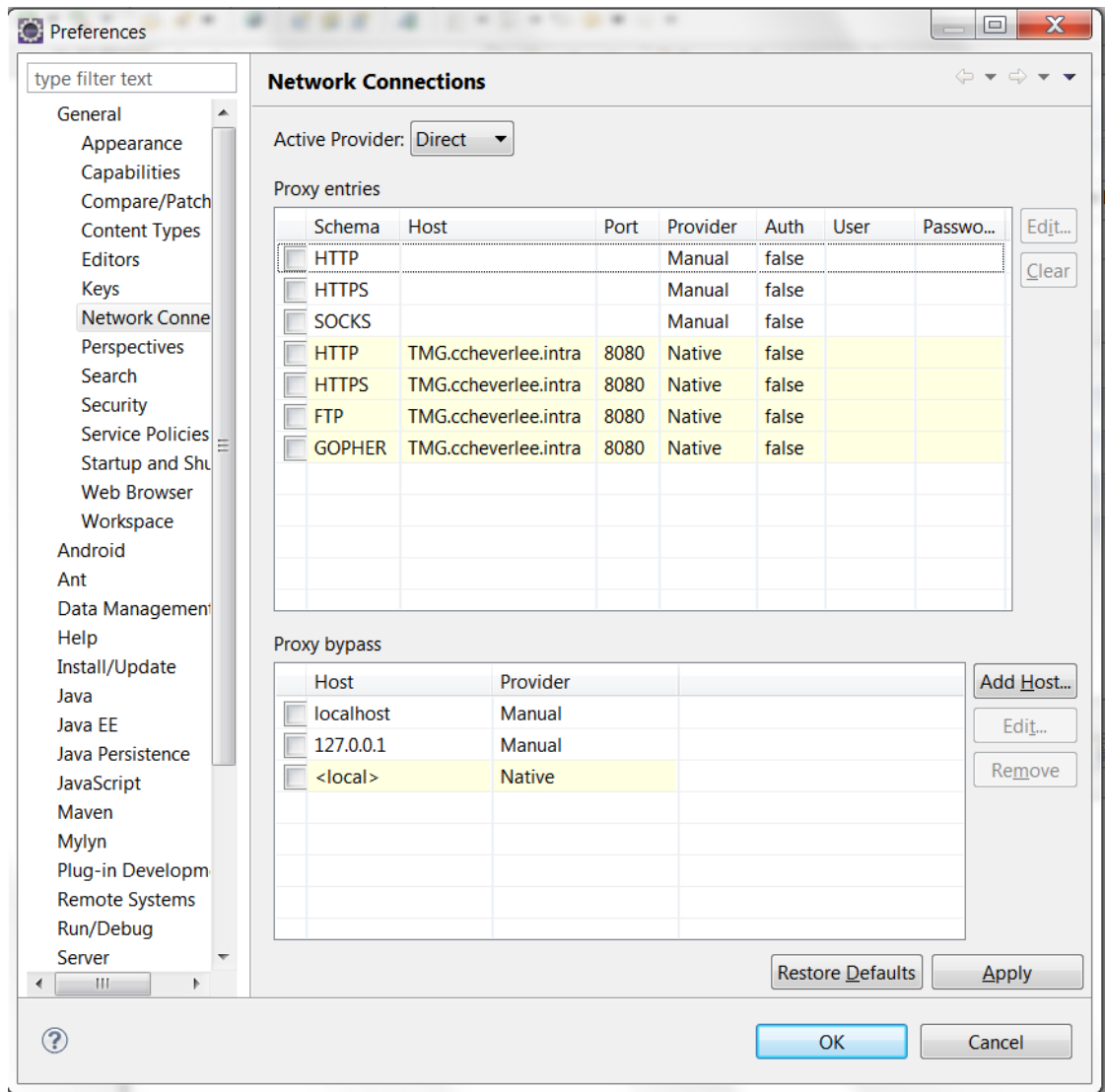
```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-framework-bom</artifactId>
      <version>4.0.1.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

In de Spring cursus zullen we gebruik maken van een Bill Of Materials.

8 ADDENDUM

8.1 Maven gebruiken achter een proxy

- Bevind je je achter een Microsoft proxy server en heb je een proxy client of TMG client op je pc, dan hoef je niets te doen en de verbinding met internet zal functioneren indien deze client correct geconfigureerd is.



- Eclipse heeft instellingen voor een proxy server. Deze vind je via het menu Window – Preferences – General – Network Connections. Er zijn reeds een aantal instellingen voorgedefiniëerd. Je klikt hier aan wat voor jou van toepassing is. In een windows omgeving met een proxy client geïnstalleerd, mag je bij Active Provider kiezen voor **Direct**, dan zorgt de Proxy client voor de authenticatie. Je overlegt best even met de instructeur/netwerkbeheerder wat de beste instellingen zijn.
 - Zit je achter een andere proxy, dan moet je dit opgeven in het Maven configuratiebestand *settings.xml*. Let op: er zijn twee versies van dit bestand! Eén bevindt zich in de Maven programmadirectory (conf) en één in de directory **•m2** die zich bevindt in je home directory. Op

Linux/Unix/Mac is deze directory verborgen.

Bij voorkeur pas je het settings bestand uit de **•m2** directory aan omdat dit bij de gebruiker hoort en niet wordt gewijzigd bij de installatie van een nieuwe versie van Maven.

Pas het bestand als volgt aan:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns=http://maven.apache.org/SETTINGS/1.0.0
          xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                    http://maven.apache.org/xsd/settings-1.0.0.xsd">
<settings>
.
.
<proxies>
  <proxy>
    <active>true</active>
    <protocol>http</protocol>
    <host>url.van.de.proxyserver</host>
    <port>8080</port>
    <username>UwGebruikersNaamVoorDeProxy</username>
    <password>UwWachtwoord</password>
    <nonProxyHosts>c1.local.be|*.onzehost.be</nonProxyHosts>
  </proxy>
</proxies>
.
.
</settings>
```

Merk op: het settings bestand bevat al voorbeelden van mogelijke configuraties, het enige wat je hoeft te doen is de code uit de commentaar te halen en de waardes aan te passen. Het is nog beter om de code uit de commentaar te kopiëren en buiten de commentaar binnen de juiste tags te plakken. Zo behoud je het originele voorbeeld.

Je kan de instellingen van meerdere proxy servers opnemen in je settings bestand. Er kan er natuurlijk maar één tegelijkertijd actief zijn. Het is mogelijk dat jouw proxy server geen authenticatie vraagt en dan kan gebruikersnaam en wachtwoord overbodig zijn. In een windows-omgeving met de proxy client software actief doe je alsof er geen proxy server is en neem je dus geen proxy configuratie op in settings.xml of plaats je de *active* tag van alle proxy servers op false.

De nonProxyHosts zijn servers die zich binnen ook achter de proxyserver bevinden en dan moeten de requests voor deze servers niet naar de proxy worden gestuurd.

Voorbeelden van nonProxyHosts zijn de Maven repository server, de subversion server of de Continuous Integration-server. Een CI-server haalt bij elke wijziging de source code op van de CVS/SVN server een voert dan een build uit van alle code, zo weet je direct dat jouw code er

voor zorgt dat er fouten optreden. Meer over CI vind je o.a. op <http://www.parleys.com/#st=5&id=2173&sl=3>

8.2 Maven-war-plugin

In de uitvoer van je compilatieproces zie je **soms** onderaan volgende foutmelding ivm web.xml.

```
C:\Users\voornaam.familienaam\workspace\cultuurhuis.naam.voornaam\target\cultuurhuis.naam.voornaam.war
[WARNING] Warning: selected war files include a WEB-INF/web.xml which will be ignored
(webxml attribute is missing from war task, or ignoreWebxml attribute is specified as 'true')
```

Doordat je vooraan in de pom.xml hebt opgegeven dat het package resultaat een war file is, wordt impliciet gebruik gemaakt van de maven-war-plugin. Op internet vind je volgende oplossing om de foutboodschap te vermijden.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>2.1.1</version>
  <configuration>
    <packagingExcludes>WEB-INF/web.xml</packagingExcludes>
    <webXml>src/main/webapp/WEB-INF/web.xml</webXml>
  </configuration>
</plugin>
```

Hierbij zeg je tegen de plugin dat deze WEB-INF/web.xml niet mee moet verpakt worden, daardoor zal de bug in de huidige versie van de plugin geen valse positieve warning meer geven. Maar je loopt wel het risico dat wanneer de bug is opgelost, je geen web.xml meer zal hebben in je war bestand.

Om er zeker van te zijn dat de maven-war-plugin het web.xml bestand toch mee neemt, geef je de locatie van web.xml expliciet mee in de *webXml* tag.

In de *webXml* tag heb je een absoluut path opgenomen. Je kan ook gebruik maken van maven properties en zo op een relatieve wijze verwijzen naar de correcte locatie. Een lijst met Maven properties vind je op:

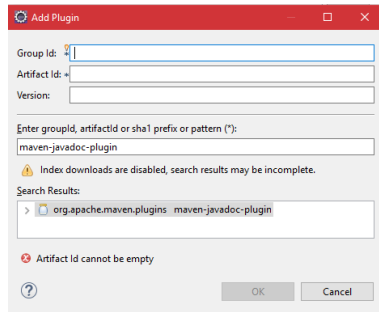
<http://docs.codehaus.org/display/MAVENUSER/MavenPropertiesGuide>. Je kan de webXml tag wijzen in: `<webXml>${basedir}/src/main/webapp/WEB-INF/web.xml</webXml>`.

8.3 Maven & Javadoc

Standaard produceert Maven geen javadoc van de code. Dit is nochtans een erg nuttige bron van documentatie. Je kan Maven vragen de javadoc te laten genereren, maar Maven heeft hiervoor de maven-javadoc-plugin nodig.

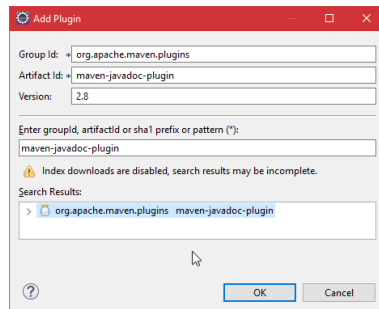
Je installeert deze plug-in als volgt:

1. Klik rechts op de POM
2. Klik in het context menu Maven – Add Plugin.



3. vul de naam of een deel van de naam in en wacht even.

4. Selecteer de plugin in de lijst van resultaten



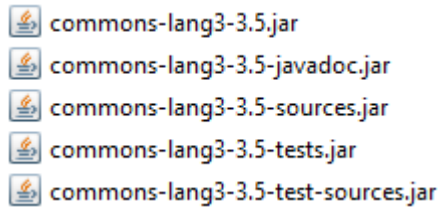
5. Klik op OK om de plug-in toe te voegen.

De plug-in bevat 10 goals:

javadoc	Genereert de Javadoc bestanden voor het project. Alle parameters van de javadoc tool zijn ondersteund.
test - javadoc	Genereert de Javadoc bestanden voor de test bestanden van het project.
aggregate	Genereert de Javadoc bestanden voor een deelproject (een aggregator project).
test - aggregate	Genereert de Javadoc bestanden voor de test bestanden van een deelproject.
jar	Creëert een JAR bestand met daarin de gegenereerde Javadoc bestanden. Deze goal wordt gebruikt bij het releasen van het project om het Javadoc artifact te bouwen. Dit artifact wordt dan geüpload naar de repository met de gecompileerde binary bestanden en het JAR bestand met de sources.
test - jar	Creëert een apart JAR bestand met daarin de gegenereerde Test Javadocs.
aggregate-jar	Creëert een apart JAR bestand met daarin de gegenereerde Javadocs van de aggregate projecten (projecten met een eigen pom die deel uitmaken van een groter project met een eigen pom)
test - aggregate-jar	Creëert een apart JAR bestand met daarin de gegenereerde Javadocs van de testen van de aggregate projecten.
fix	Is een interactieve goal die de Javadoc documentatie en tags voor de Java files herstelt.
test - fix	Is een interactieve goal die de Javadoc documentatie en tags voor de test Java files herstelt.

In deze stap zal je de javadoc genereren en verpakken in een JAR bestand. Dit jar bestand kan je uiteindelijk installeren op de lokale repository of deployen naar de bedrijfs repository. De collega die dan onze code downloadt van de repository, beschikt onmiddellijk over de documentatie van ons project. Dit is vooral gebruikelijk bij open-source projecten.

Zo herkennen we in de [Apache commons lang](#) onder andere de jar met de



gecompileerde code, een jar met de source en een jar met de documentatie.

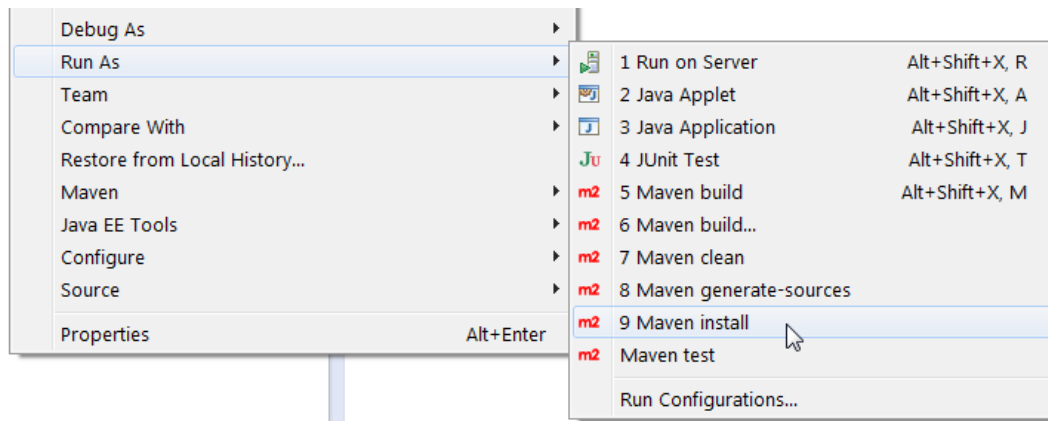
Pas de configuratie van de plug-in aan in xml weergave:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-javadoc-plugin</artifactId>
  <version>2.8</version>
  <configuration>
    <outputDirectory>
      ${project.build.directory}/apidocs
    </outputDirectory>
  </configuration>
  <executions>
    <execution>
      <id>create-javadoc-jar</id>
      <phase>package</phase>
      <goals>
        <goal>jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

In de lijst van executions zit één execution waarbij je de jar goal van de maven-javadoc-plugin laat uitvoeren tijdens de package fase van de default lifecycle.

Je vindt meer informatie over de plug-in op <http://maven.apache.org/plugins/maven-javadoc-plugin/>.

Wanneer je nu aan Eclipse om je project uit te voeren als Maven install zal Maven de documentatie genereren.



Wanneer je het commando "Run As ""Maven package"" uitvoert op je project en je kijkt onder de target directory dan vind je daar de gegenereerde documentatie.

8.3.1.1 Meer over de Maven site plug-ins

We hebben de reporting tag al opgenomen in pom.xml.

Vanuit plug-ins die je configureert in deze tag laat je rapporten genereren met documentatie over het project.

De **maven-jxr-plugin** genereert documentatie die lijkt op javadoc maar de sourcecode bevat links naar andere classes binnen het project.

Met de **maven-surefire-report-plugin** integreer je de JUnit resultaten in een HTML pagina binnen de projectdocumentatie.

De **cobertura-maven-plugin** kan je gebruiken om na te gaan hoeveel procent van onze code gedekt is door JUnit tests. Dit noemt men de code-coverage. Het rapport geeft met een balkje weer hoeveel procent van een package of class getest is. Het percentage ongeteste code staat in een rood balkje, de geteste in een groen balkje. Je kunt doorklikken tot in de source code van de class. De geteste lijnen code hebben een groene achtergrond, de ongeteste lijnen code een rode achtergrond.

De **maven-checkstyle-plugin** geeft fouten weer tegen de regels die afgesproken zijn voor het coderen van het programma. Het opstellen van die regels is waarschijnlijk een job voor de Java architect van het team of het bedrijf. Er zijn reeds 4 gedefinieerde stijlen ter beschikking. Als je geen stijl hebt opgegeven wordt de **sun_checks.xml** gebruikt. Uitgebreide documentatie over deze plug-in en de standaard Sun stijl vind je op <http://checkstyle.sourceforge.net/checks.html>.

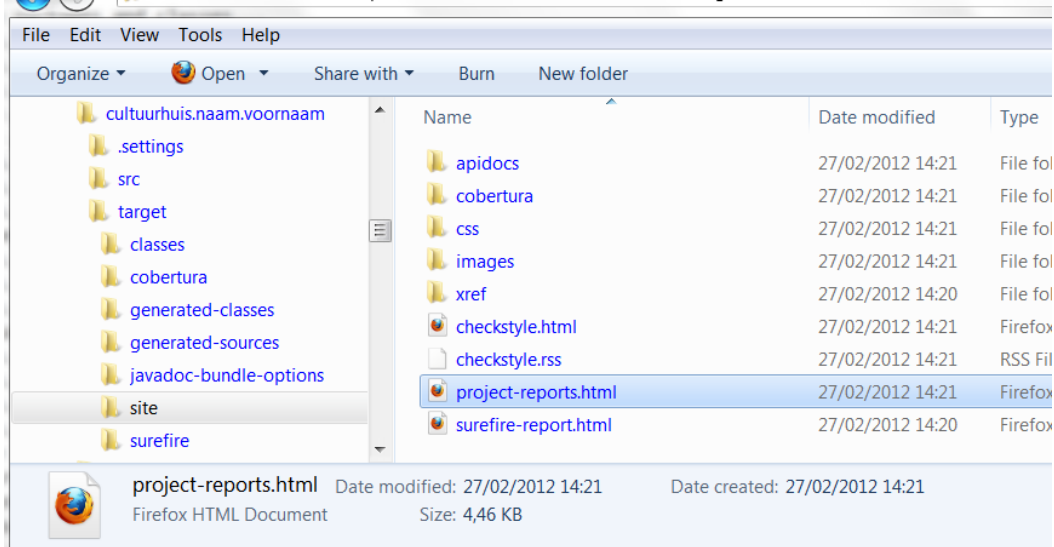
Je laat de documentatie als volgt genereren:

1. Klik rechts op de naam van je project
2. Klik op Run As
3. Klik op 7 Maven build...
4. In het invoervak van Goals typ je site
5. Klik onderaan in de dialog op de knop Run

In het console venster verschijnt de vooruitgangsinformatie.

Wanneer Eclipse klaar is, kan je met de file explorer kijken in de project directory. De *target* directory bevat een directory *site*. Deze bevat de gegenereerde HTML

code. Open met je browser de file *project-reports.html*.



Dit is de hoofdpagina van je project documentatie. Je vindt hier de linken terug naar de verschillende onderdelen van de gegenereerde documentatie.

Generated Reports

This document provides an overview of the various reports that are automatically generated by **Maven**. Each report is briefly described below.

Document	Description
Test JavaDocs	Test JavaDoc API documentation.
JavaDocs	JavaDoc API documentation.
Source Xref	HTML based, cross-reference version of Java source code.
Test Source Xref	HTML based, cross-reference version of Java test source code.
Surefire Report	Report on the test results of the project.
Cobertura Test Coverage	Cobertura Test Coverage Report.
Checkstyle	Report on coding style conventions.

Copyright © 2012. All Rights Reserved.

Wanneer je project nog geen classes en test classes bevat zal bovenstaande pagina veel beperker zijn. Aan het einde van deze cursus kotm een oefening waarbij de een volledig rapport zal kunnen genereren.

Zowel de Javadocs van de testclasses als van de source herken je van de API-documentatie van java zelf.

All Classes

DatumExceptionTest
DatumTest

be.vdab.util.DatumTest

public class DatumTest
extends Object

Author:

frank.roelants

Constructor Summary

Constructors

Constructor and Description
DatumTest()

Method Summary

Methods

Modifier and Type	Method and Description
void	test_28_februari_geen_schrikkeljaar_100()
void	test_28_februari_geen_schrikkeljaar()
void	test_28_februari_schrikkeljaar_400()
void	test_28_februari_schrikkeljaar()

De Source Xref documentatie toont de sources op een gelijkaardige wijze als de API documentatie, maar hier krijg je alle details i.p.v. een overzicht.

All Classes

Packages

[be.vdab.example](#)
[be.vdab.util](#)

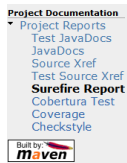
All Classes

[Datum](#)
[DatumException](#)
[ExampleServlet](#)

```
1 public Datum(int dag, int maand, int jaar) throws DatumException{
2     valideerDatum(dag, maand, jaar);
3     this.dag=dag;
4     this.maand=maand;
5     this.jaar=jaar;
6 }
7
8 public int getDag(){
9     return dag;
10 }
11
12 public int getMaand(){
13     return maand;
14 }
15
16 public int getJaar(){
17     return jaar;
18 }
19
20 private void valideerDatum(int dag, int maand, int jaar) throws DatumException{
21     int dvm[]={31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 31, 30, 31};
22     boolean goedeDatum = jaar>=1583&&jaar<=4099&&
23         maand>0&&maand<13&&
24         dag>0&&
25         dag<=(dvm[maand-1]+((maand==2&&
26             (jaar%4==0&&jaar%100!=0||jaar%400==0)?1:0));
27
28     if ( !goedeDatum ) {
29         throw new DatumException("Datum is fout");
30     }
31 }
32
33 @Override
34 public String toString(){
35     return String.format("%02d/%02d/%d", getDag(),getMaand(),getJaar());
36 }
37
38 @Override
39 public boolean equals(Object obj){
40     boolean returnValue=false;
41     if( obj!=null &&
42         this.getClass() == obj.getClass()){
43         //
44     }
45 }
```

Via dit rapport kan je een project grasduinen door in de source te klikken op de links.

In het surefire rapport krijg je een overzicht van de uitgevoerde JUnit testen.



Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
46	44	0	0	4.348%	0.161

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
be.vdab.util	46	44	0	0	4.348%	0.161

In het cobertura rapport zie je hoeveel percent van je code gedekt is door JUnit tests. Je kan in het rapport de source code bekijken. De achtergrondkleur geeft aan of de code gebruikt werd in een test (groen) of niet (rood).

Wanneer je in een method een andere method oproept is de tweede method groen gekleurd in de test. Maar is deze dan ook getest? Het antwoord is waarschijnlijk neen. Wanneer je methods een selectie bevat, dan kan je nagaan of je programma in alle takken is geweest, maar het is nog steeds aan jou als programmeur om voldoende testen te schrijven die alle waardes voldoende testen.

Packages

[All](#)
[be.vdab.example](#)
[be.vdab.util](#)

Coverage Report - be.vdab.util

Package /	# Classes	Line Coverage	Branch Coverage	Complexity
be.vdab.util	2	25% 8/32	0% 0/32	2,071
Classes in this Package /		Line Coverage	Branch Coverage	Complexity
Datum		0% 0/24	0% 0/32	2,5
DatumException		100% 8/8	N/A N/A	1

Classes in this File	Line Coverage	Branch Coverage
DatumException	100% 8/8	N/A N/A

```
1
2 package be.vdab.util;
3
4 public class DatumException extends Exception {
5     public DatumException(Throwable cause){
6         super(cause);
7     }
8     public DatumException(String message, Throwable cause){
9         super(message, cause);
10    }
11    public DatumException(String message){
12        super(message);
13    }
14    public DatumException(){
```

Het checkstyle rapport toon de fouten tegen de coding conventions, welke mogelijk kunnen leiden tot bugs. Het fine-tunen van dit rapport is echt nodig, maar valt volledig buiten het bestek van deze cursus. Je kan in het rapport zien dat je een fout krijgt op lijn 9 van de servlet omdat de javadoc commentaarlijn niet werd afgesloten met een punt. Elders krijg je een fout omdat er geen spatie voor én achter een = teken staat. In het rapport kan je dus zien hoe nauwgezet je te werk ging.

Details

be/vdab/example/ExampleServlet.java

Violation	Message	Line
✖	Missing package-info.java file.	0
✖	First sentence should end with a period.	9
✖	Line contains a tab character.	13
✖	Missing a Javadoc comment.	13
✖	Line has trailing spaces.	14
✖	Line contains a tab character.	23
✖	Line is longer than 80 characters.	24
✖	Line contains a tab character.	24
✖	Line contains a tab character.	25
✖	Line is longer than 80 characters.	26
✖	Line contains a tab character.	26
✖	Parameter request should be final.	26
✖	Expected @param tag for 'request'.	26
✖	Parameter response should be final.	26
✖	Expected @param tag for 'response'.	26
✖	Expected @throws tag for 'ServletException'.	26
✖	Expected @throws tag for 'IOException'.	26

8.4 Meer informatie

Wanneer je meer informatie zoekt over Maven kan je die onder meer vinden op de website van Apache Maven zelf <https://maven.apache.org/index.html> en in één van de vier gratis e-books van Sonatype: <http://www.sonatype.com/Support/Books>. Je kan de boeken downloaden in pdf formaat in ruil voor je e-mail adres en wat informatie over je werkgever of je kan ze online lezen.

Oefening

Wanneer je tot nu toe alleen de cursus hebt gelezen zonder de stappen uit te proberen is nu het moment gekomen om de Maven plug-in in Eclipse te installeren.

Zet een web-app op met de naam “MijnProject”.

Voorzie de applicatie van een *index.jsp*. Deze pagina bevat een link naar *resultaat.htm*.

De link *resultaat.htm* is gemapt op een servlet, die een object van het immutable type *be.vdab.valueobjects.Adres* maakt en dit laat tonen in *resultaat.jsp*.

Adres bevat een straat, huisnummer, busnummer, postcode en gemeente.

Wanneer straat, huisnummer, postcode of gemeente niet ingevuld zijn, wordt een *IllegalArgumentException* gegooid.

Voorzie je servlet en je valueobject van javadoc commentaar.

Voorzie deze class van een testclass.

Laat je programma uitvoeren en genereer de site documentatie van je project zoals hiervoor beschreven.

9 PROBLEMEN?

Wanneer je problemen hebt met Maven dan is het steeds een goed idee om rechts te klikken op je project en in het menu een Maven - Update project te vragen. Hiermee synchroniseert Eclipse de eclipse project settings met de inhoud van de POM.

Krijg je nog steeds fouten dan kan je de .m2 directory in je home folder leeg te maken en daarna een Maven clean en daarna een Maven install uit te voeren.

10 OEFENING

Installeer de Maven plugin in Eclipse., indien dit nog niet gebeurde.

Zet een web-app op met de naam “MijnProject”.

Voorzie de applicatie van een index.jsp, deze pagina bevat een link naar resultaat.htm.

De link resultaat.htm is gemapt op een servlet dit een object van het immutable type `be.vdab.valueobjects.Adres` maakt en dit laat tonen in resultaat.jsp.

Adres bevat een straat, huisnummer, busnummer, postcode en gemeente.

Wanneer straat, huisnummer, postcode of gemeente niet ingevuld zijn wordt een `IllegalArgumentException` gegooit.

Voorzie je servlet en je valueobject van javadoc commentaar.

Voorzie deze class een testclass.

Laat je programma uitvoeren en genereer de site documentie van je project zoals hier voor beschreven.

11 COLOFON

Sectorverantwoordelijke: Ortaire Uyttersprot

Cursusverantwoordelijke: Jean Smits

Medewerkers: Frank Roelants

Versie: 15/06/2016

Nummer dotatielijst: