Aanwinsten -bouwen

Deze tekst beschrijft het compileren en testen van de programmatuur en het maken van een Debian package.

Stap 0 - Controleer de vereisten

Stap 1 - Uitchecken, compileren en testen Aanwinsten

```
$ cd ..
$ git clone git@gitlab.uvt.nl:lis-asm/aanwinsten.git
$ cd aanwinsten
$ mvn versions:use-latest-versions -DallowMajorUpdates=false -DallowMinorUpdates=true -Dincludes=nl.uvt.commons
$ mvn versions:commit
$ mvn clean test
```

Met de 'mvn versions:*'-commando's wordt de dependency voor uvt-commons in de pom.xml bijgewerkt naar de laatste versie in de lokale repository.

Stap 2 - Compileren Java-package voor lokaal testen

```
$ mvn package
```

Er staat in directory `target` nu een bestand `aanwinsten-<versie>-jar-with-dependencies.jar`

Deze kan worden gestart met commando:

```
$ java -cp target/aanwinsten-<versie>-jar-with-dependencies.jar \
  nl.uvt.aanwinsten.CLI <commando>
```

Voor gebruik zie de betreffende tekst.

Stap 3 - Maken Debian package

```
$ ./build-debian-package.sh
```

Het package staat na afloop in directory `package` als bestand `aanwinsten_<versie>_all.deb`

Nieuwe versie

De versie van het programma staat in drie bestanden, en moet dus ook in drie bestanden worden aangepast wanneer er een nieuwe versie moet komen. Dat is een beetje onhandig, maar de investering om tot een mooie oplossing te komen win je niet snel terug. Pas deze bestanden aan voor een nieuwe versie. De plaats wijst zichzelf:

- pom.xml (voor Maven, de bouw-tool)
- control (t.b.v. het Debian package)
- build-debian-package (om juiste versienummer in het .deb-bestand te krijgen)

De scripts `maak-lijsten` en `maak-adhoc-lijsten` refereren naar de jar met versie en moeten ook worden aangepast wanneer daar gebruik van wordt gemaakt.

Maken site met onder andere uitkomsten statische code analyses

Door het gebruik van het volgende commando

```
$ mvn site
```

wordt een statische site gemaakt die veel informatie bevat over het project. Open daarvoor bestand target/site/index.html in een browser.

Onder Project Information mogelijk interessante informatie, bijvoorbeeld bij 'Source Repository'. Onder Project Reports vind je de JavaDocs en rapportages van de statische code analyses. Bij die laatste staat soms commentaar dat onterecht is omdat de statische code analyseerders niet altijd even goed met Lombok-annotaties om kunnen gaan.

Configureren Eclipse in verband met Lombok

De programmatuur maakt gebruik van library Lombok. Lombok neemt veel van het saaie repetitieve werk in Java weg. In een IDE als Eclipse geven de Lombok-constructies problemen voor de syntax-checker. Met behulp van een installatie-commando kan je zorgen dat Eclipse de programmatuur eerst door Lombok laat interpreteren en daarna pas bepaalt of de syntax okee is.

Stop vooraf Eclipse!

Spoor de Lombok-jar op in de lokale repository (`~/.m2/repository`). De versie van Lombok bij schrijven van deze handleiding is `1.16.6`. Controleer bestand `commons/commons-parent/pom.xml` voor variabele `org.lombokproject.version`.

Installeer Lombok in Eclipse. Het bestand is bijvoorbeeld `~/.m2/repository/org/projectlombok/lombok/1.16.6/lombok-1.16.6.jar`. Vervang de versie door de juiste.

```
$ java -jar ~/.m2/repository/org/projectlombok/lombok/1.16.6/lombok-1.16.6.jar
```

Omdat bij mij de Eclipse-programmatuur eigendom is van gebruiker 'root' en groep 'eclipse', kan een 'gewone' gebruiker daarin niet schrijven in de Eclipse-directory. In dat geval in Linux sudo gebruiken:

```
$ sudo java -jar ~/.m2/repository/org/projectlombok/lombok/1.16.6/lombok-1.16.6.jar
```

Lombok opent een dialoog-scherm. Geef daarin op waar Eclipse staat; de directory waar bestand `eclipse.ini` staat. Bij mij was dat directory `/opt/eclipse.mars`. Het programma plaatst bestand `lombok.jar` in de opgegeven Eclipse-directory en past bestand `eclipse.ini` in die directory aan.

Na starten Eclipse zouden de Lombok-constructie niet meer tot fouten moeten leiden. Misschien moet je even een Project | Clean... doen (menu Eclipse).